Fault Management at JPL: Past, Present and Future

ADCSS 2011

John Day & Michel Ingham
Jet Propulsion Laboratory, California Institute of Technology

Outline

- **Position Statement**
- **Fundamentals**
- **Past Experience and Lessons Learned**
- **Current State of Practice**
- **Future Evolution**
- **Summary**

POSITION STATEMENT

Position Statement

At JPL and across the spacecraft engineering community, the development of robust FDIR* capabilities has been *more of an "art" than a "science"*.

We posit that there is significant benefit to be gleaned from applying *greater rigor and a more systematic approach* to FDIR system development, and that the burgeoning field of *Model-Based Systems Engineering* can provide useful techniques and tools to help us in this endeavour.

* Note: In this package, we will use the somewhat more general terms "Fault Management" and "Fault Protection". There are subtle distinctions between each of these terms, which we can discuss offline if there is interest.

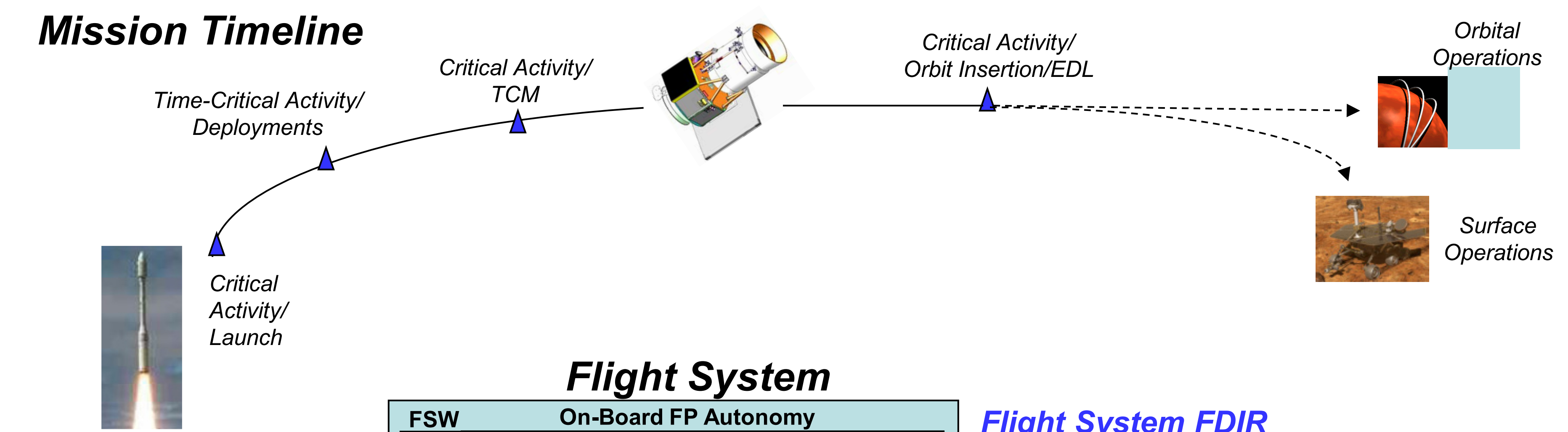
FUNDAMENTALS

What is Fault Protection?

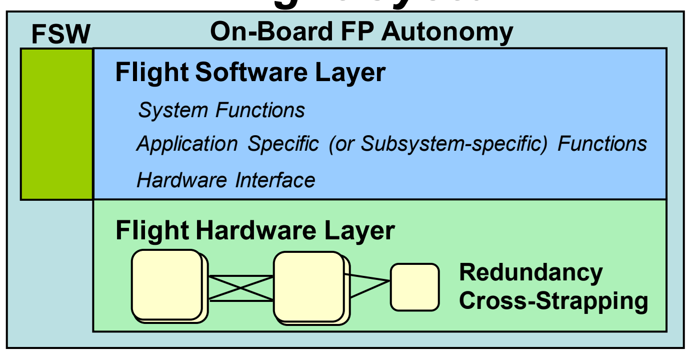
- **As used and applied at JPL, Fault Protection is <u>both</u>:**
  - A specific SE discipline (similar to EEIS or mission planning), whose activities should be separately scheduled and tracked, and
  - The elements of a system that address off-nominal behavior
- **Focused on the flight system, Fault Protection includes**
  - Flight system fault detection and response
  - Failure diagnosis and recovery
  - Ground contingency planning and action
- **"Fault Management" is becoming the preferred term within NASA**
  - Fault Protection is functionally equivalent to "Fault Management*", but with a flight system bias

** Per current draft of NASA-HDBK-1002, "Fault Management Handbook"; however, the definition remains in work…*

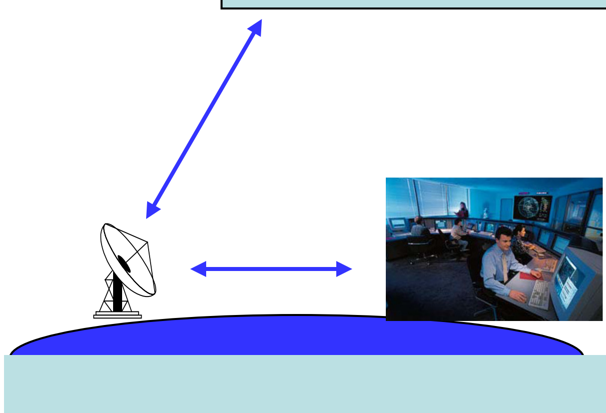
Fault Protection Context



Flight System



Flight System FDIR
* FSW Layers
* Hardware Layer



Ground System

Ground FDIR
* Monitor/Trend
* Diagnosis/Recovery
* Contingency Plans / Procedures
* Test-bed/Simulation

Fault Protection Scope

Fault Protection

Fault Avoidance

In-Flight Examples include:
1) Robust Design Features
- Simple Design (e.g. fixed solar array or HGA)
- conservative design practices and performance margins
2) On-Board Autonomy
- (e.g. attitude constraint checking).
3) Post-Launch Operations Processes and Procedures
- These include the use of Flight Rules and executing commands on a spacecraft simulation test-bed.

Fault Tolerance

Flight System Fault Tolerance Design Strategies include:
1) Graceful Degradation
2) Application of Redundancy
- physical
- functional
3) Fault Containment Regions
- limit propagation
4) FDIR

Fault Masking

For Example:
1) EDAC
2) FPGA TMR
3) Active Redundancy

FDIR

Ground-based FDIR

For Example:
1) Telemetry Alarm Checking
2) Telemetry Trending & Analysis
3) Contingency Plans & Procedures

Flight HW FDIR

For Example:
1) OV/UV Detect Circuits
2) Watchdog Timers
3) On-Chip Built-in-Tests
4) Memory scrubbing

Flight SW FDIR

For Example:
1) Sensor Threshold / Persistence
2) Diagnosis / Response
3) Re-plan & Continue Activity

PAST EXPERIENCE

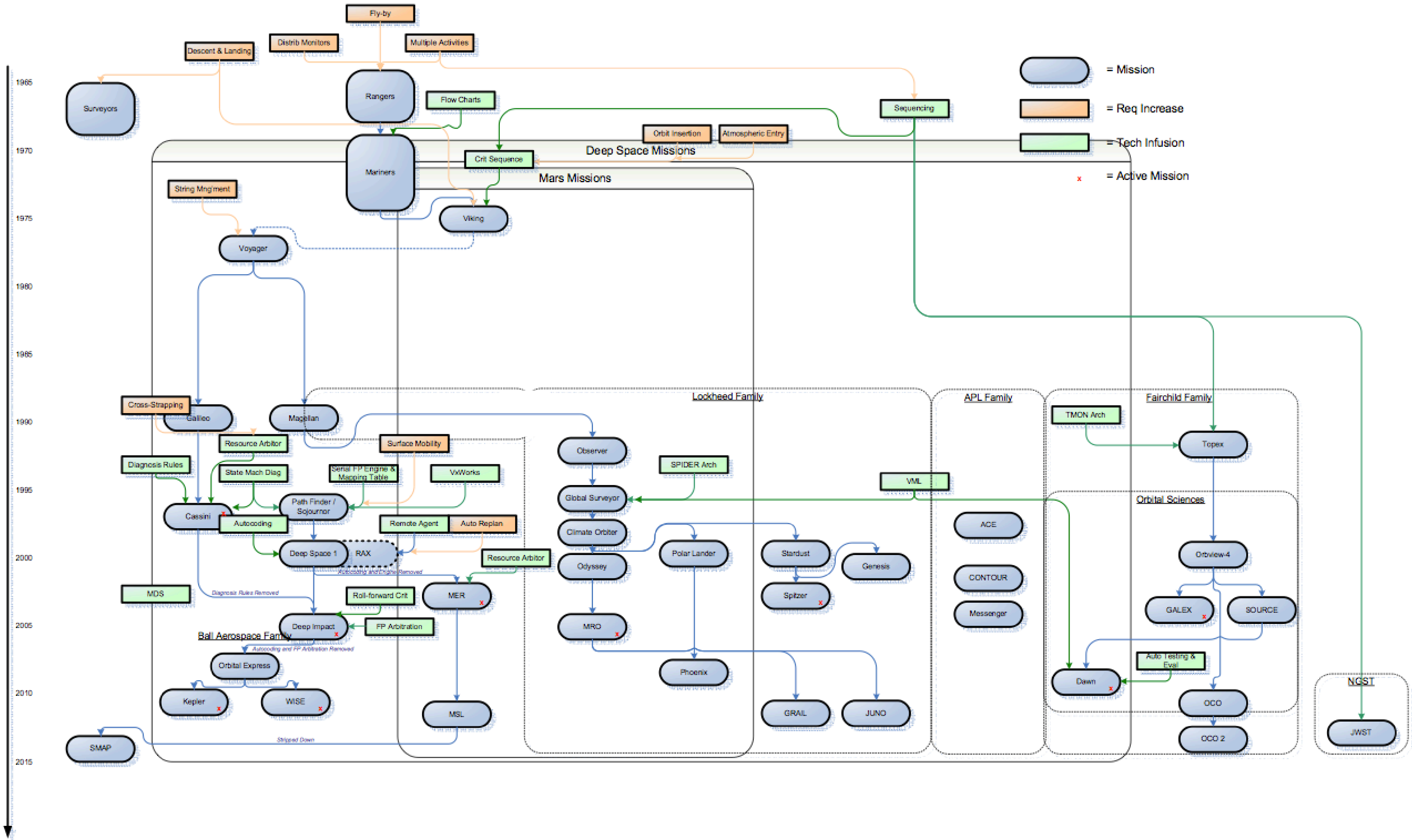
Missions and Capabilities

- **The set of missions historically flown by JPL has led to the development of robust autonomous FP capabilities**
  - Short time-to-criticality, long light-times, limited contact with operations teams, no maintenance opportunities, time-critical events

- **FP capability fielded on Viking and Voyager, gradually increasing in scale to significant levels of complexity and autonomy**
  - Cassini SOI is a good example of autonomous FP capability
  - JPL FP designs and processes formed by experience and lessons learned (some painfully)

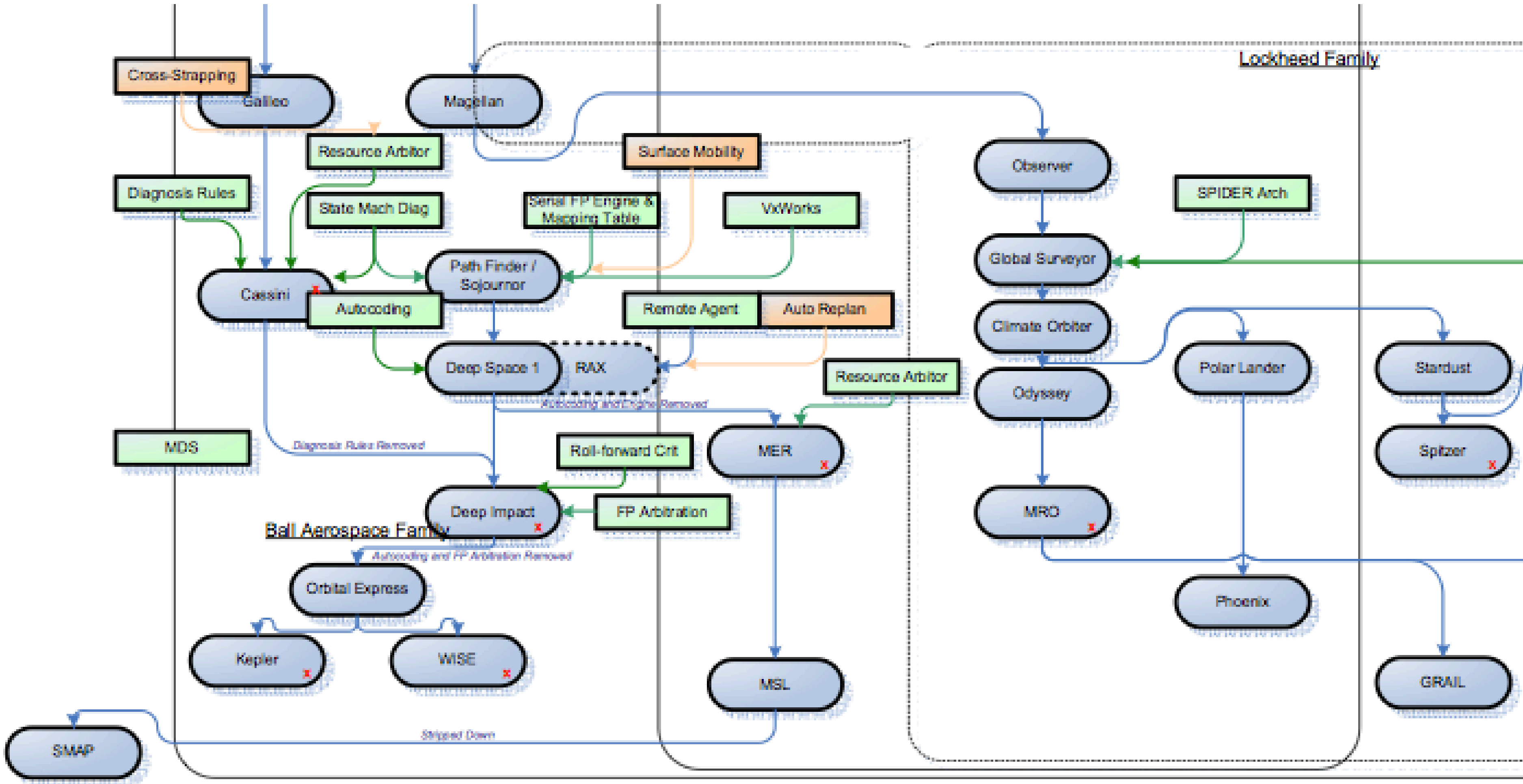
- **MSL represents the most complex FP system JPL has built, with 1097 monitors and 38 responses**

Fault Protection "Family Tree*"



*See reference [16], "OCE FSW Complexity Study, Appendix F.

FP "Family Tree" – Detail



Typical Constraints and Driving Requirements

- **Operate with Limited Ground contact**
  - *Extended periods with no planned contact* *(1 to 4 weeks)*
  - *Planned contact periods may be short* *(1 to 2 hours)*
  - *Ground may not show for planned contacts* *(5% to 10%)*
  - *Large one-way light times* *(minutes to hours)*
  - *Low downlink data rates* *(10 to 40 bps)*
- **Protect fragile elements of systems**
- **Leverage existing flight system components**
- **Protect/complete critical activities**
  - Orbit insertion, entry/descent/landing, irreversible deployments
- **Long mission life**
  - Survive *without maintenance* for primary missions lasting 5-11 years
- **Harsh environments**
  - TID of 100 krad to 4 mrad

In-Flight Experience with Fault Protection

- **JPL missions have suffered relatively few permanent faults**
  - *Flight hardware for deep space missions has to be (and has been) very reliable*
- **Fault protection activity during our missions has been most commonly caused by:**
  - *Operator errors*
  - *Fundamental design flaws, including software design flaws*
  - *False alarms due to unnecessarily tight thresholds*
  - *Unforeseen transient behavior due to interactions and/or variations in the operating environment, SEUs, etc.*
- **Many examples where fault protection responded appropriately to transient behavior that was unexpected**
  - *Galileo (1990 - 1995): Despun Power Bus reset caused by debris shorts*
  - *Magellan (1990 - 1992): Software flaw that caused heartbeat termination*
  - *Cassini (1993): Attitude estimator transient during backup Star Tracker checkout*
  - *MER Spirit Rover (2005): Potato-sized rock jammed in right rear wheel*
  - *Dawn (2008): Cosmic ray upset of attitude control electronics*
  - *Kepler (2009): Undervoltage due to unexpected power interactions at launch*

PRESENT

Characteristics of JPL FP Approach

- **Single-failure tolerance (SFT)**
  - No single point of failure will result in loss of mission
  - For some missions, waived in part or whole (e.g., single-string)

- **Limited use of reliability data**
  - JPL does not use reliability estimates as a basis for meeting single-failure tolerance requirements
  - Reliability estimates used for lifetime calculations
  - Reliability estimates used as supporting rationale in SFT waivers

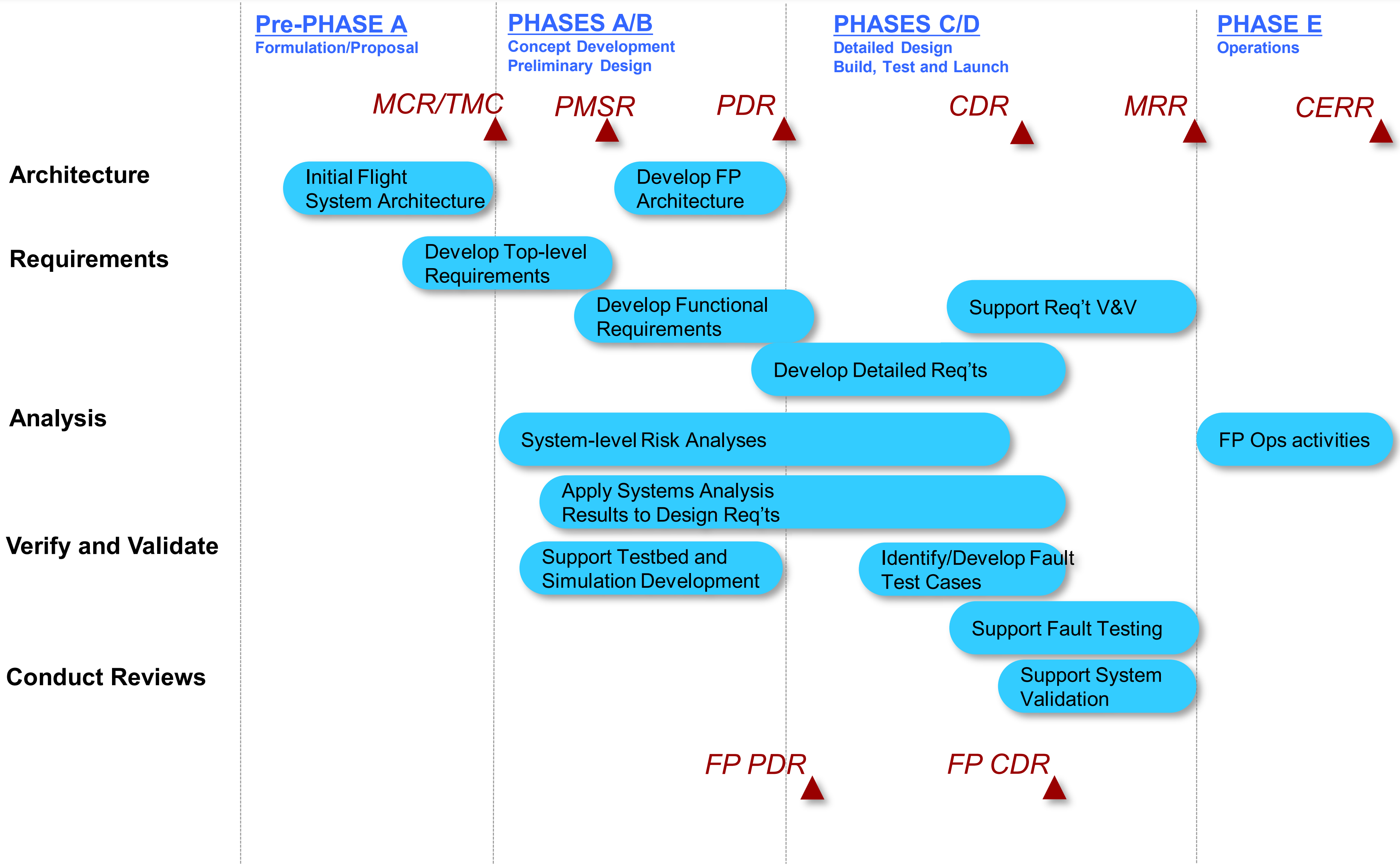
- **Maintain failure tolerance after first failure**
  - Clear temporary failures
  - Maintain failure tolerance in safing modes
  - Robustness to multiple orthogonal failures

Fault Protection System Engineering

- **On JPL flight projects, Fault Protection is a broad-based systems engineering task, and includes components of:**
  - **Mission Engineering**
    - Timeline, Nominal, Critical and Time-Critical Activities
  - **Project System Engineering**
    - Systems Architecture
  - **Flight System Engineering**
    - Failure Analysis
    - Requirement/Design Flow-down to FSW, Subsystem SE, Reliability
    - Design, Test, and Operation of On-Board autonomous Fault Detection, Isolation, and Response logic responsible for maintaining vehicle health and safety.
      - Hardware Redundancy is often included
  - **Mission Operations**
    - Contingency Planning and Anomaly Resolution
    - Flight System Data Analysis and trending, state tracking, simulation
  - **Mission Assurance**
    - Reliability Analysis, Parts Qualification, Environments etc.

- **The FP effort is often managed like a ‘spacecraft subsystem’.**
  - Reviews, budget/schedule (WBS), specific work products
  - Keeps effort from being lost or or mismanaged

FP Across the Project Lifecycle



**See reference [9], "Fault Protection System Engineering: Tasks and Products Across the Project Lifecycle" for more detail.*

Implementation Challenges

- **Show quantitative benefits to support engineering trades**
  - Developing approaches to show value of additional HW and SW
  - Especially - assessing value of applying HW redundancy

- **Accurately estimate and control costs**
  - Better define products and processes, and process metrics

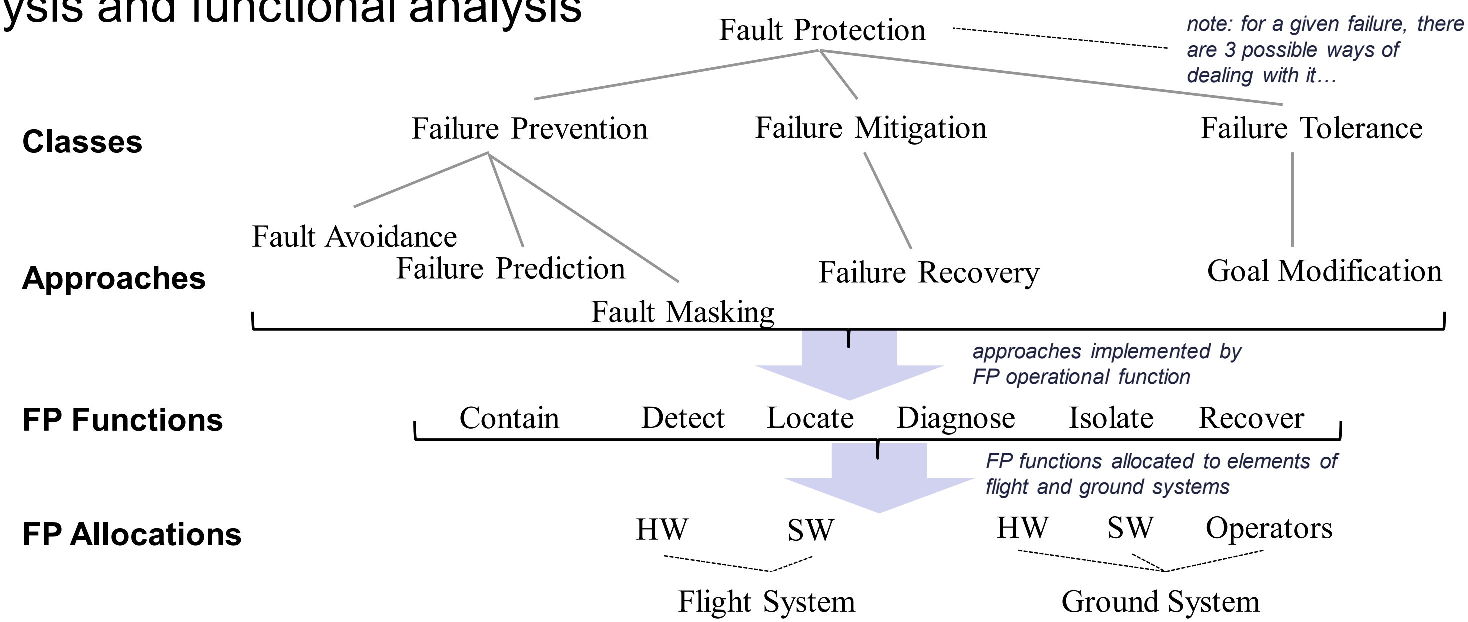
- **Perform adequate V&V**
  - Large failure space makes comprehensive testing infeasible
  - Working on tools and approaches to better verify and validate

- **Write relevant, decomposable requirements**
  - Needs to be more than “Do FP”
  - Better integration with SE requirements process

FUTURE EVOLUTION

Directions of Current Research (1)

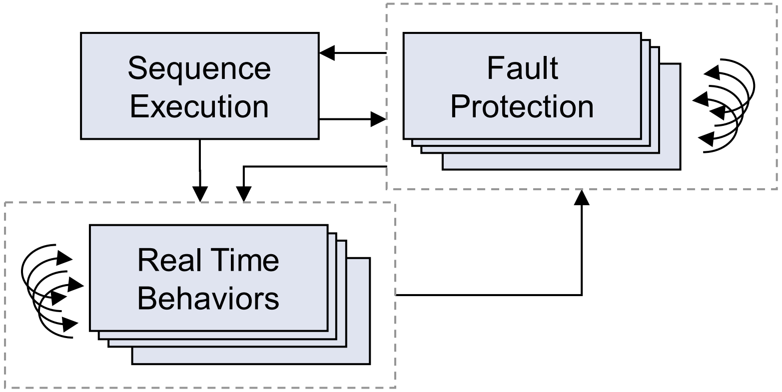
- **Advancing the "Science" of Fault Management**
  - Formalization of concepts and terminology
  - Development of unified Theory of FM, leveraging prior work on state analysis and functional analysis

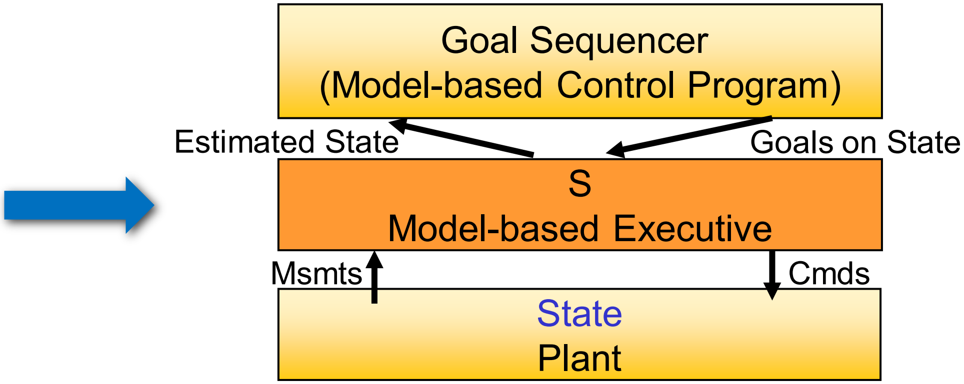


- **Improved Fault Management design process**
  - Integration of FM design into "mainline" Systems Engineering activities
  - Application of Model-Based Engineering (MBE) techniques to document FM design and enable difficult (or previously impossible) analyses

Directions of Current Research (2)

- **Resilient system architectures**
  - Development of system architectures that are inherently capable of fault avoidance, tolerance and recovery, rather than fault protection architecture as a “bolt-on” to nominal execution architecture.
    - Integration of fault protection within the nominal control loop
    - Continued migration of “cognizance” from operators to spacecraft



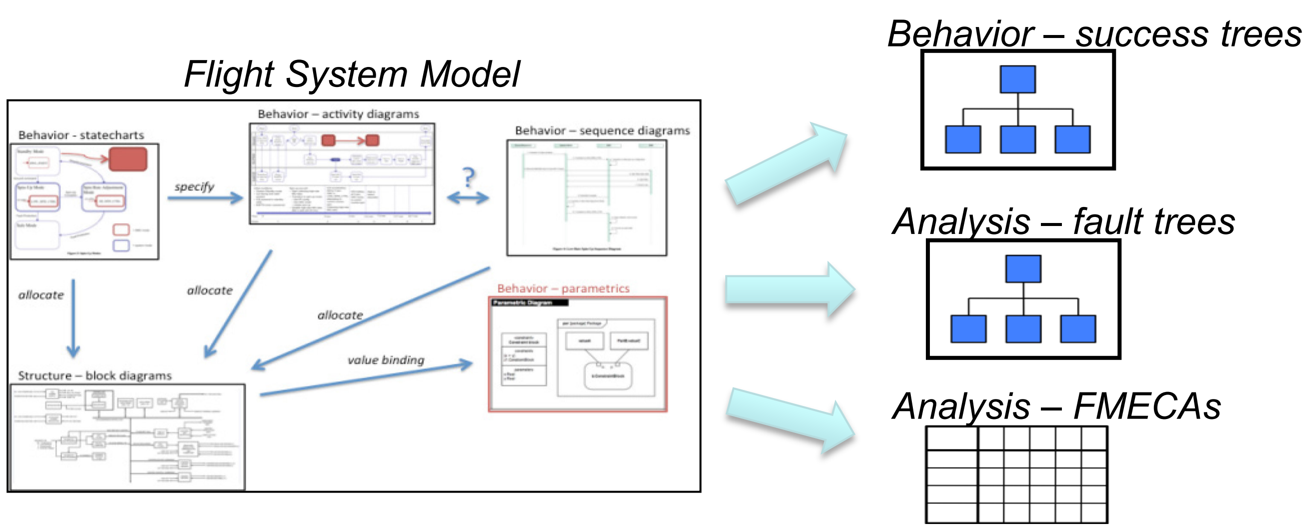


- **Advanced diagnosis & recovery algorithms**
  - Leverages recent advances in model-based reasoning, hybrid (discrete/continuous) system modeling, discrete-event systems and Integrated System Health Management (ISHM) communities
  - Challenges: modeling expressivity, coherent integration of multiple representations and techniques, and scalability to large-scale systems

Directions of Current Research (3)

- **Formal verification methods, verifiable software, autocoding**
  - MBE and model-driven software development provide greater opportunity for formal V&V techniques and automated code generation
  - Building up “libraries” of code-generation patterns for use in future missions

- **Fault management design environments**
  - Development of model transformation technologies to integrate general-purpose MBE languages (e.g., SysML) & tools with FM-specific design environments (e.g., TEAMS, SAFIRE)
  - Eventual automation of generation of FM analysis artifacts (e.g., FT, FMECA)



Summary

Past:

- **JPL has a long history of developing, deploying and operating effective Fault Management capabilities on its spacecraft**
- **Our FM capabilities have evolved as our missions have become increasingly ambitious and complex, but this evolution was not rigorously "architected" over time**

Present:

- **JPL Fault Protection philosophies and goals are relatively straightforward and generally consistent from project to project**
- **FP engineers end up knowing how the Flight System really works (and how it doesn't work), better than anyone**

Future:

- **JPL is working with the FM Community to advance the state of the art and practice, to enable future classes of missions**
  - Formalize theory, improve and standardize approaches and processes, develop tools (move from an "art" to a science)
  - Increase our collective ability to field safe and reliable systems
  - Enable formulation and development of more complex/capable systems

Opportunities to Continue the Discussion

- **2nd NASA Fault Management Workshop (New Orleans, Louisiana; April/May 2012)**
  – By invitation only
  – Contact Dr. Lorraine Fesq for more information: lorraine.m.fesq@jpl.nasa.gov

- **Fault Management sessions at AIAA Infotech@Aerospace 2012 (Anaheim, California; June 18-21, 2012)**
  – Call for Papers: **www.aiaa.org/events/I@A**
  – Abstracts due November 22, 2011

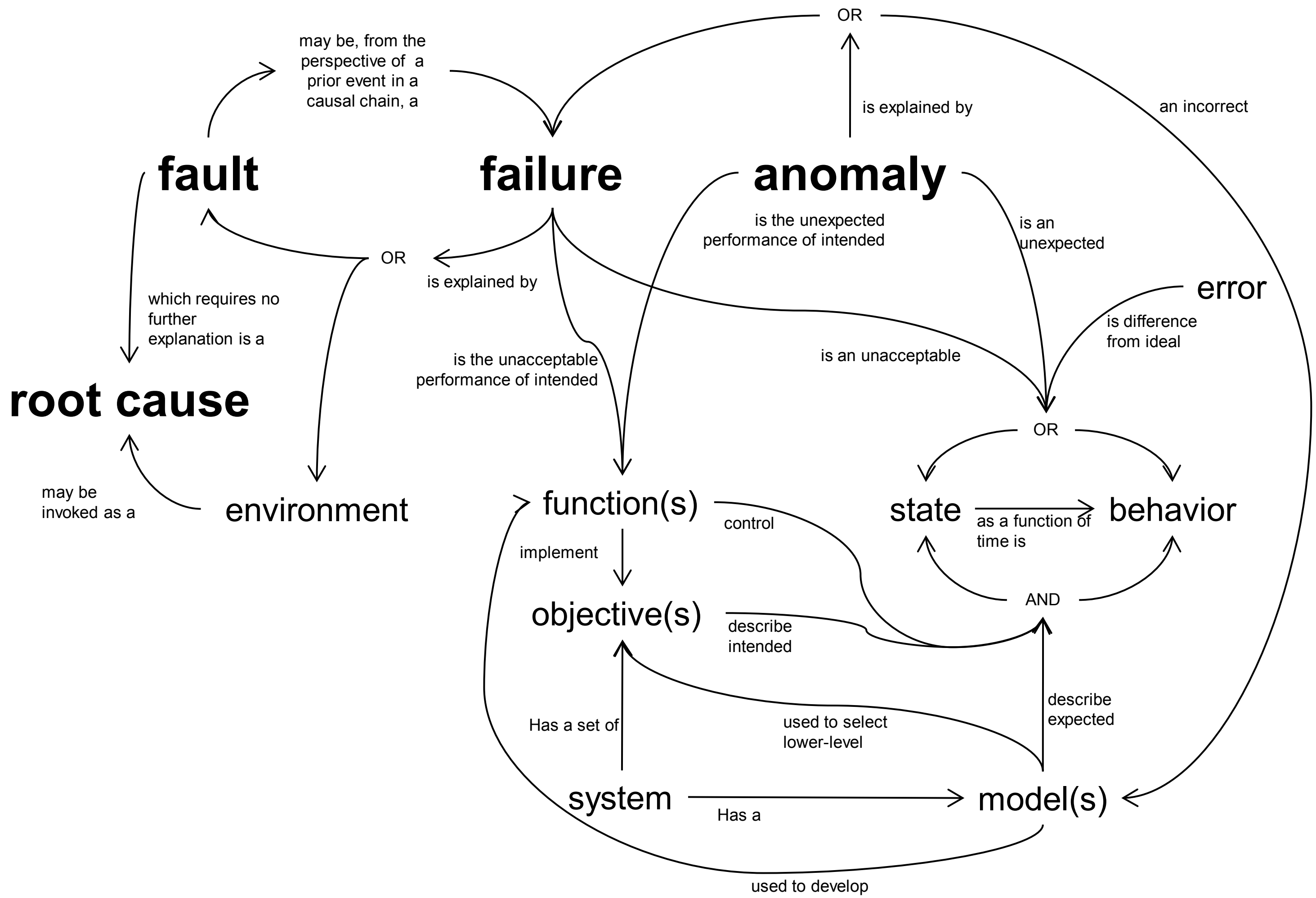
BACKUP

NASA SMD FM HB: Core and System Terms

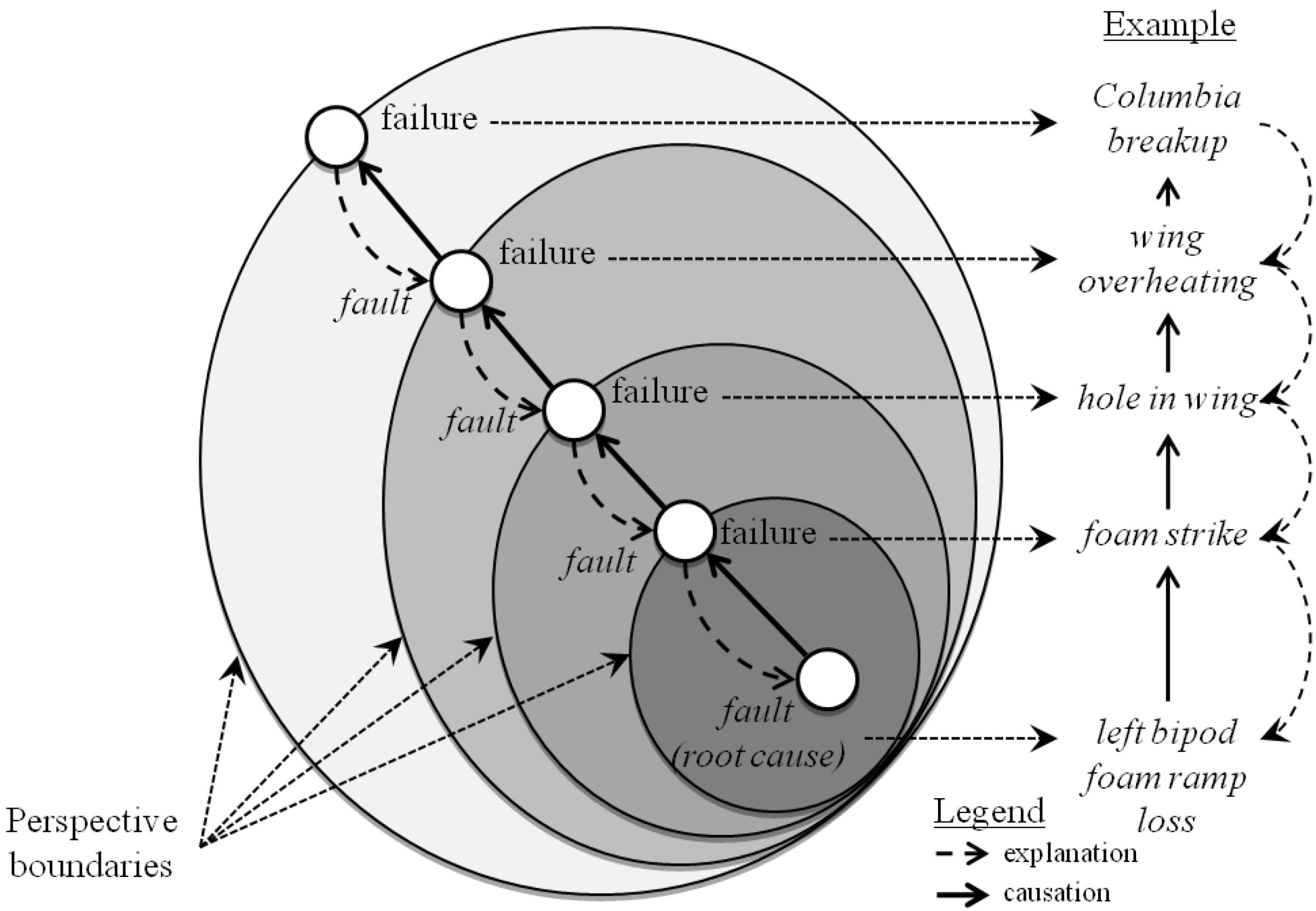
- **Core Terms**
  - ***Degradation***: The decreased performance of intended *function*.
  - ***Anomaly***: The unexpected performance of intended *function*.
  - ***Failure***: The unacceptable performance of intended *function*.
  - ***Fault***: A physical or logical cause, which explains a *failure*.
  - ***Root Cause***: In the chain of events leading to a *failure*, the first *fault* or environmental cause used to explain the existence of the *failure*

- **System Terms**
  - **System**: A combination of interacting elements organized to achieve one or more stated purposes.
  - **State**: The value of a set of physical or logical state variables at a specified point in time.
  - **Behavior**: The temporal evolution of a *state*.
  - **Function**: The process that transforms an input *state* to an intended output *state*.
  - **Control Error**: The deviation between the *estimated state* and the ideal intended *state*.
  - **Nominal**: The *state* of the *system* when the output *state* vector matches the intentions of the designer and/or operator.
  - **Expectation**: The most likely predicted *state* or *behavior*.

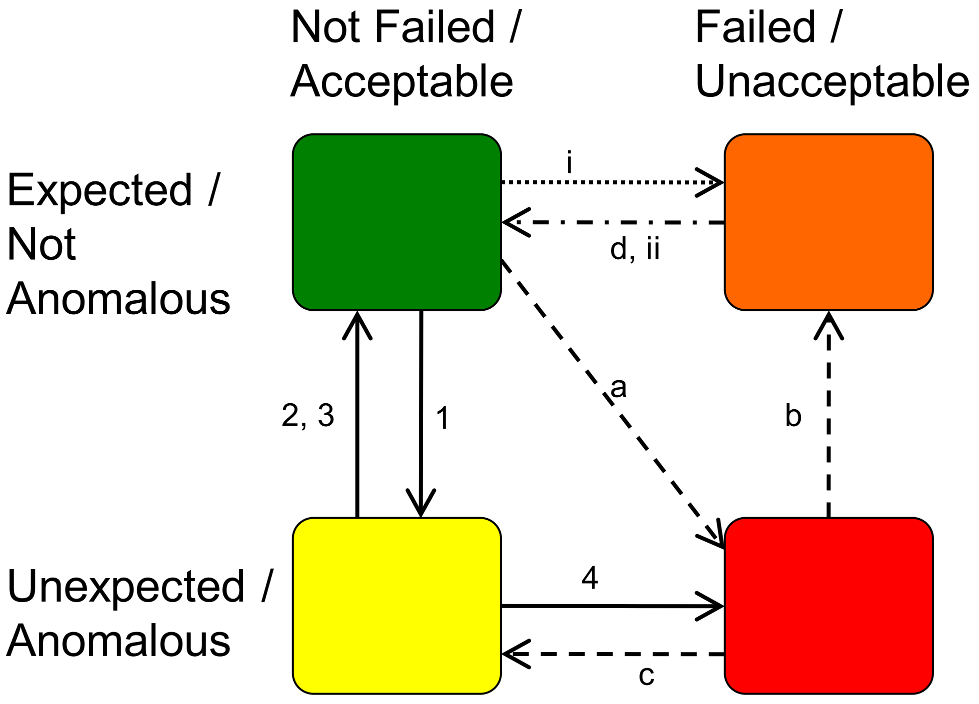
Terminology Concept Diagram



Fault – Failure Recursion



Progression of Anomalous/Failed States



Anomaly, no Failure

1) current value of state reaches an unexpected value
2) review of system data indicates that model/expectation is invalid, and state is expected (expectations changed) [e.g., noise in RF link due to un-modeled effect]

- model reviewed and parameters adjusted until model predicts current behavior (e.g., if RWA unhealthy, will have larger attitude errors)
- review of system data indicates that this is an unacceptable value (indicative of a failure; the goal is adjusted)

Anomaly, with Failure

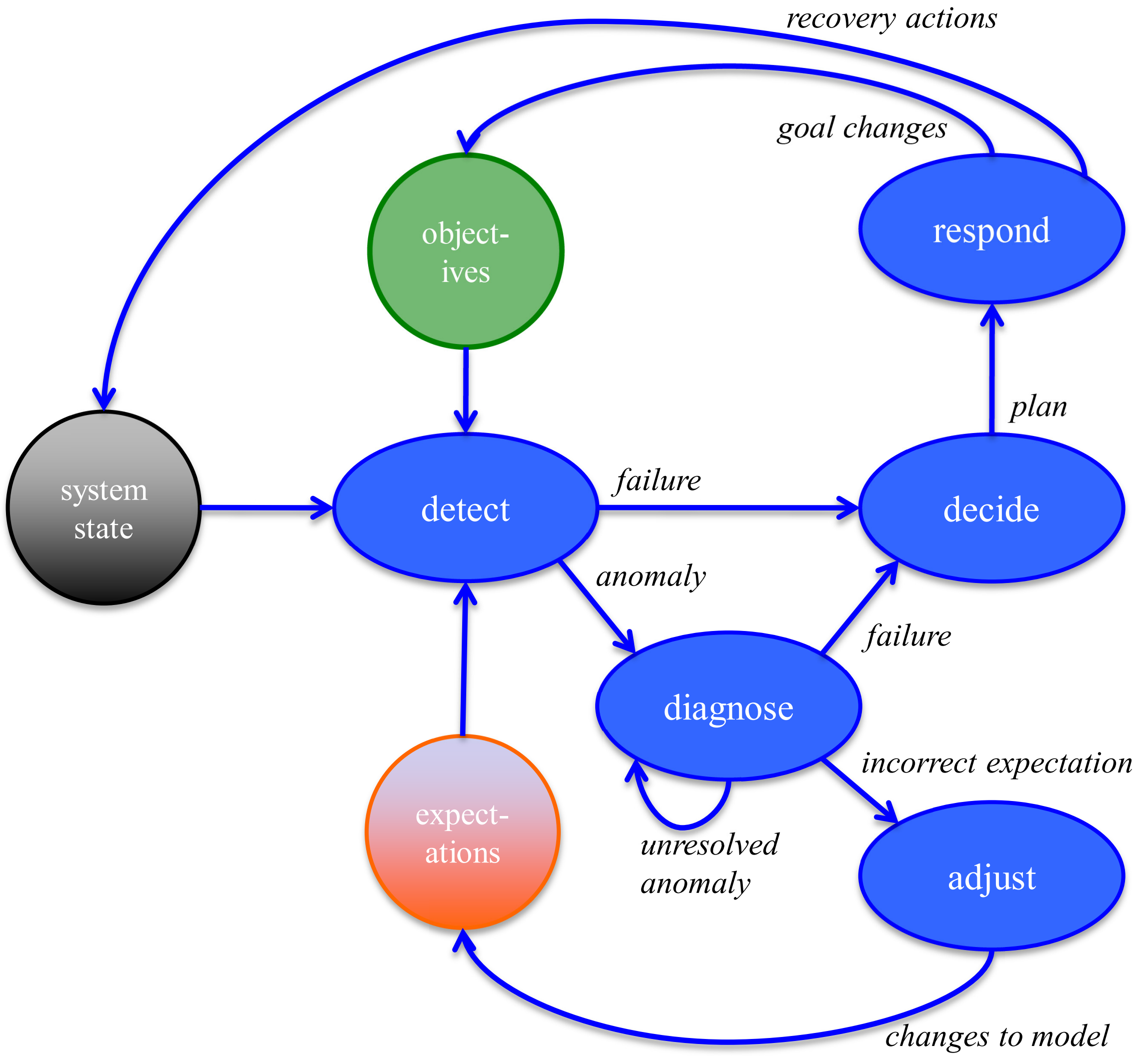
a) current value of state unexpectedly reaches an unacceptable value
b) model reviewed and parameters adjusted until model predicts current behavior (e.g., if IMU1 unhealthy, will have attitude failure)

- review of system data indicates that model/expectation is invalid, and state is acceptable (expectations changed)
- recover intended functionality by restoring state to acceptable value and/or changing functional goal

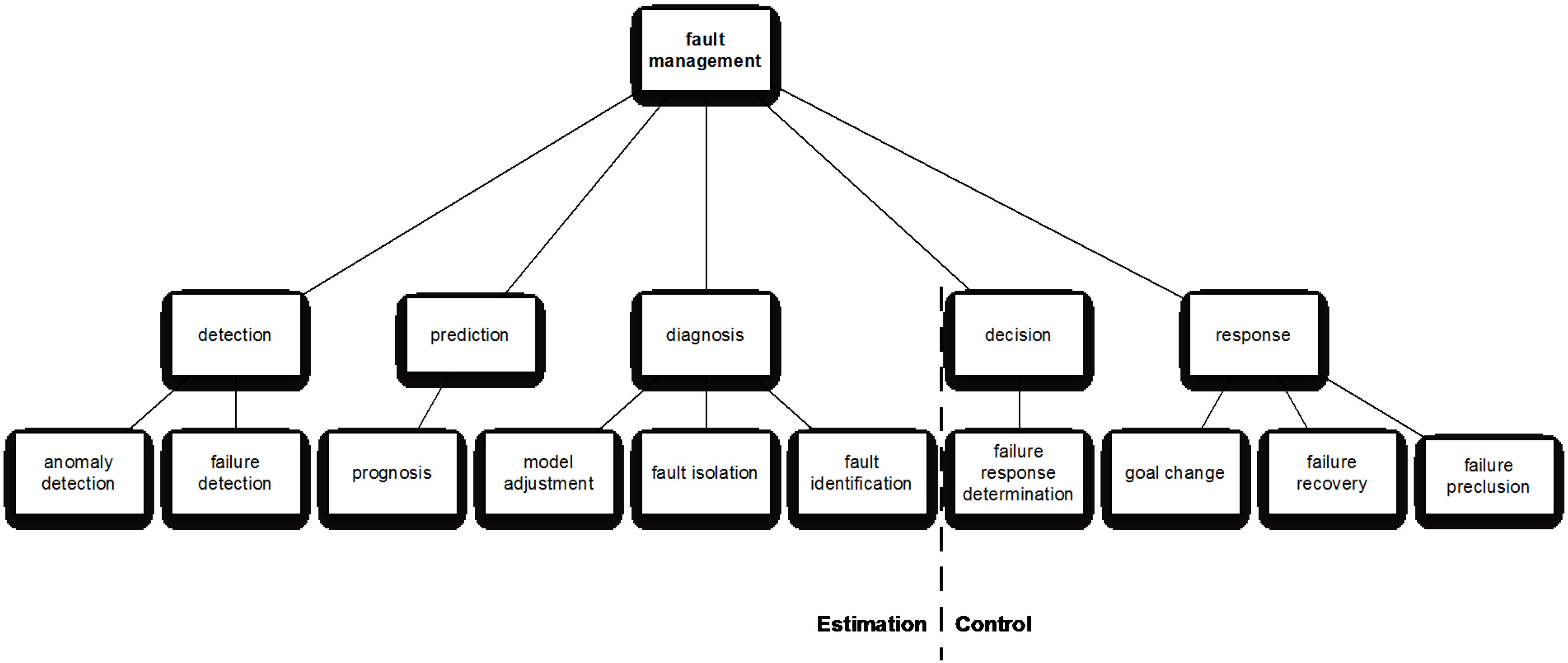
Failure, no Anomaly

i. expected condition results in failure
ii. recover intended functionality by restoring state to acceptable value and/or changing functional goal

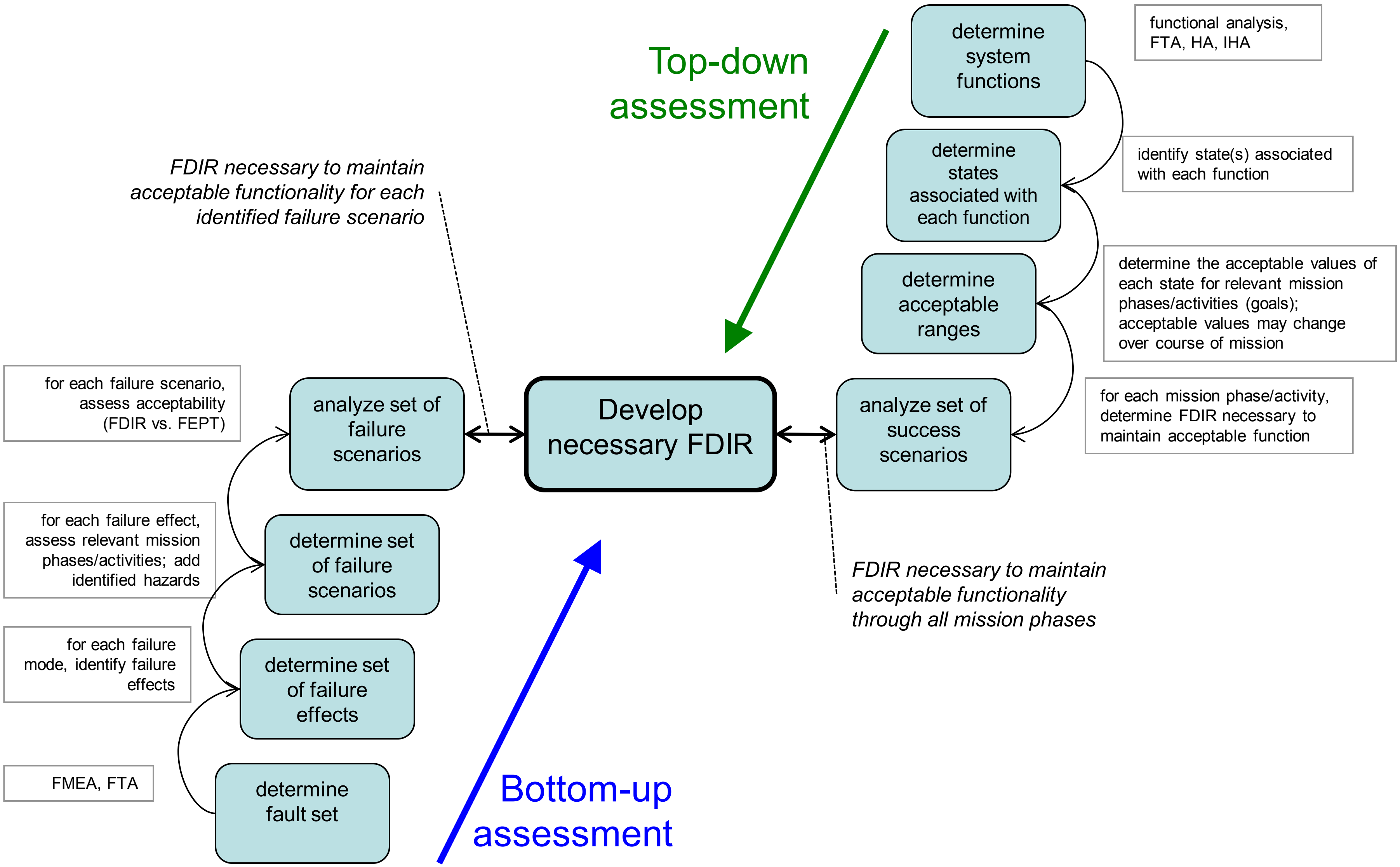
Simplified Fault Management Loop



FM Functions

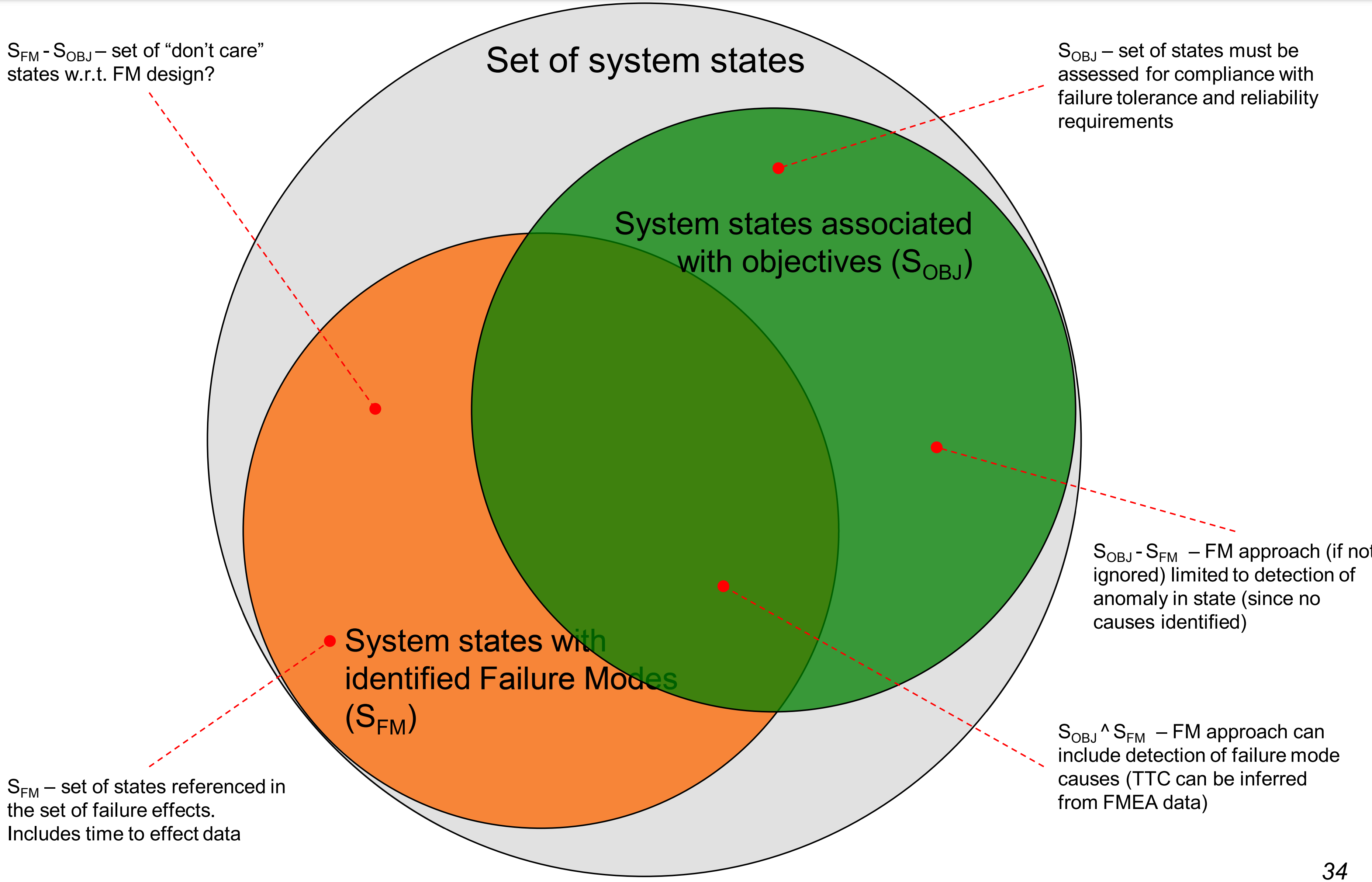


FM Completeness: Requires Top-Down and Bottom-up Analyses



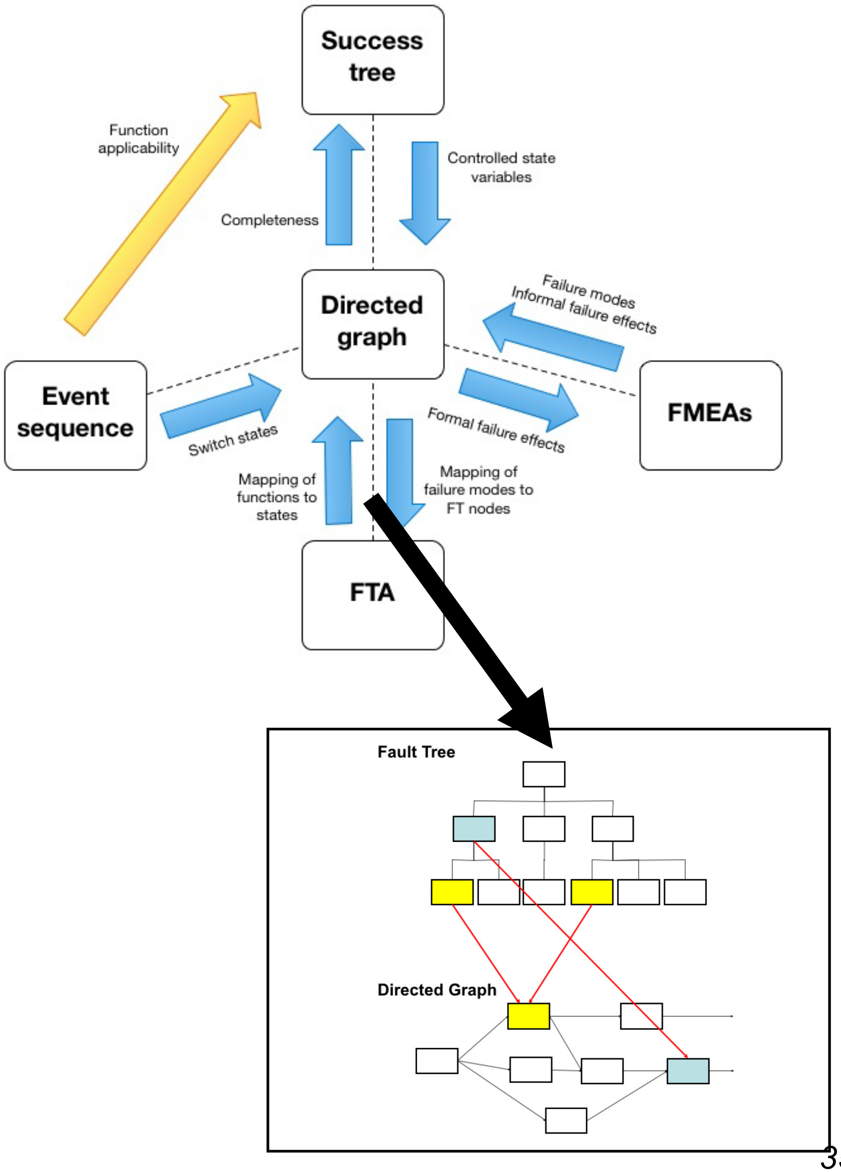
list of local fault responses

System States

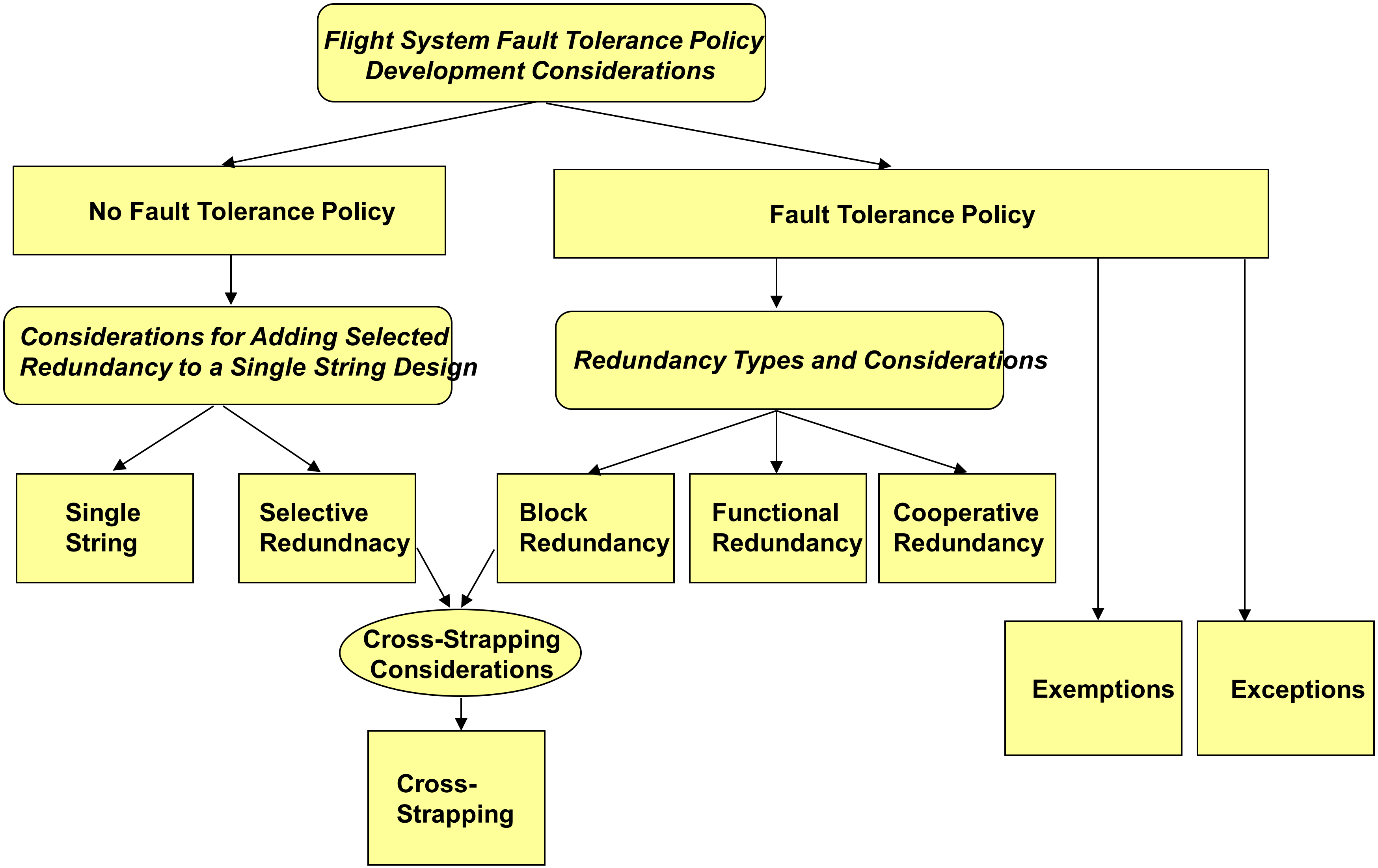


Relevant Representations and Relationships

- **Success Trees**
  - Represent system functions and functional decomposition
  - Conditions for success; "light" side
- **Fault Trees**
  - Represent system functions and paths to failure of top event
  - Conditions for failure; "dark" side
- **Directed graphs**
  - Represent components and connections/interfaces
  - Modeling of physical and logical connections enables formal modeling of failure effect propagation
- **Failure Modes and Effects Analyses (FMEA)**
  - Description of the failure modes (mechanisms) and the immediate failure effect
  - Modeled failure effect propagation enables formal and complete development of all failure effects
- **Event Sequences**
  - Describes system functionality as a function of time
  - Provides "triggers" to enable/disable elements of directed graph representation
- **State Machines** (Not Shown)
  - Necessary to assess sequencing of system states, both nominal and off-nominal



Redundancy and Cross-strapping Guidelines



Fault Protection Components:
Flight System FDIR / H/W Layer

- **Key JPL design practice requires definition of Fault Containment Regions (FCRs):**
  - *"A fault containment region (FCR) is a segment of the system, the design of which is such that faults internal to the fault containment region do not propagate and cause irreversible damage beyond the limits of the fault containment region.* ***Note:*** *Fault propagation can be both direct/obvious (e.g. damage, disabling) and indirect/subtle (e.g. contention, interference)."*

- **Fault containment boundaries in the flight equipment are always drawn around each of the following [8]:**

  1 - any redundant elements (either functional or block redundant).

  2 - any non-critical functions or equipment (e.g. any item where it's function is not required for mission success, such as engineering telemetry, instruments etc.).

  3 - any protective functions or equipment that are conditionally needed, (e.g. OV/OC protect)

  4 - any functional area or equipment the project requires to be fault tolerant.

  5 - any functional area or equipment the projects requires fault containment for development risk (e.g. difficult to replace, long-lead, unique, or costly items are prime candidates for fault containment boundaries for development risk.)

- **FCRs are also important in Single String Designs**

Fault Management Architectures[1]

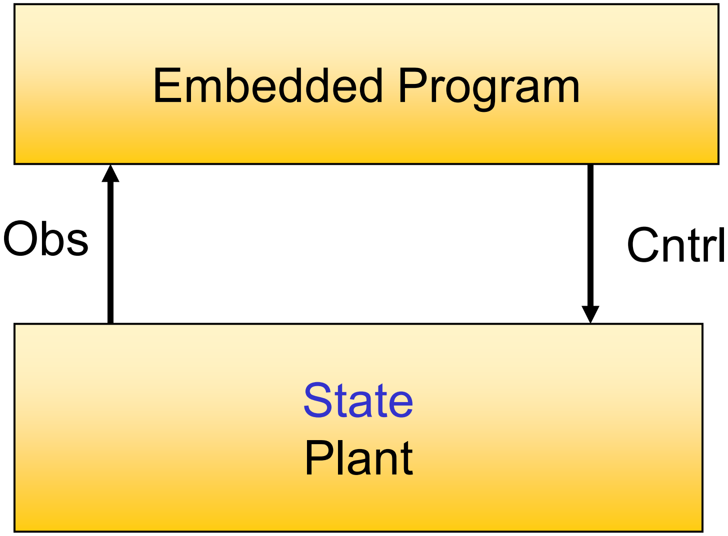
	Practitioner:	*Lockheed*	*Goddard*	*Orbital*		*APL*	*JPL*					*Ball Aerospace*
	Family:	*"Spider" + VML*	*TMON*			*Rule-Based*	*Parallel State Machines*	*Smart Sequences*	*Local Software Logic*	*Reusable Fault Protection Framework*		
	Missions:	*MRO / Phoenix / MPL / MO*		*Dawn*	*GALEX*	*New Horizons / Messenger*	*Cassini AACS*	*Cassini CDS*	*MER*	*Pathfinder /DS-1*	*Deep Impact*	*Kepler / WISE / Orbital Express*
Fault Response	*Deployment*	Local	Central	Central	Central	Central	Central	Central	Local	Central	Central	Central
	Thread Control	TBD	Parallel	Parallel	Parallel	Parallel	Parallel	Parallel	Parallel	Serial	Serial	Serial
	Interaction Management	State Checks	Enables /Disables in FP Seqs	Enables /Disables in FP Seqs	Enables /Disables in FP Seqs	State Checks	State Checks	TBD	Resource Contention Checks	None	Resource Contention Checks	TBD
	Behavior Selection	Table-Defined Tiers	Table-Defined Tiers	Table-Defined Tiers	Table-Defined Tiers	Macro Logic	State Machines withTiers	Table-Defined Tiers	TBD	State Machines withTiers	State Machines withTiers	TBD
	Behavior Pacing	TBD	Monitor Persistence	Monitor Persistence	Monitor Persistence	TBD	Response Delay Logic	TBD	TBD	TBD	Response Delay Logic	TBD
	Primary Means of Command Execution	Sequenced	Sequenced	Sequenced	Sequenced	Sequenced (Macros)	In-line	Sequenced	In-line	In-line	Sequenced	Sequenced
	Responsiveness to System State	Via Sequence Syntax	N/A	N/A	N/A	Via Rule syntax	Via Response Code	Via Sequence Syntax	Via Response Code	Via Response Code	Via Response Code	TBD

1

Model-based Programs Reason about State

Embedded programs interact with the system's sensors/actuators:

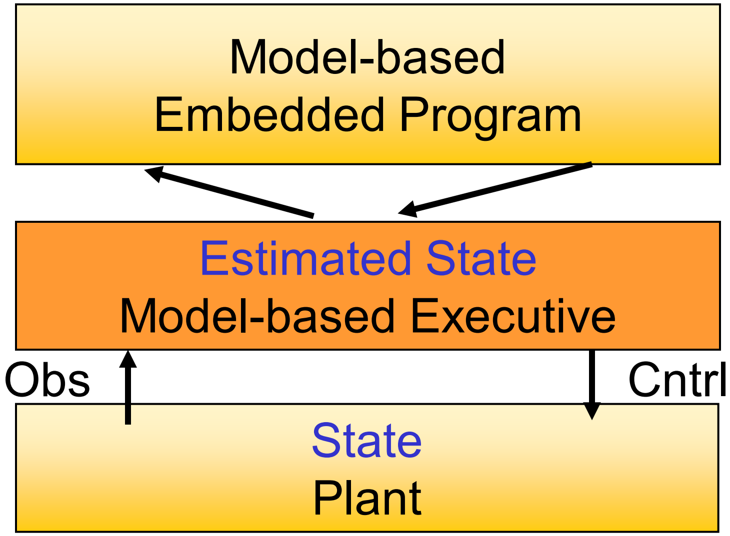
- Read sensors
- Set actuators



Programmers must reason through interactions between state and sensors/actuators.

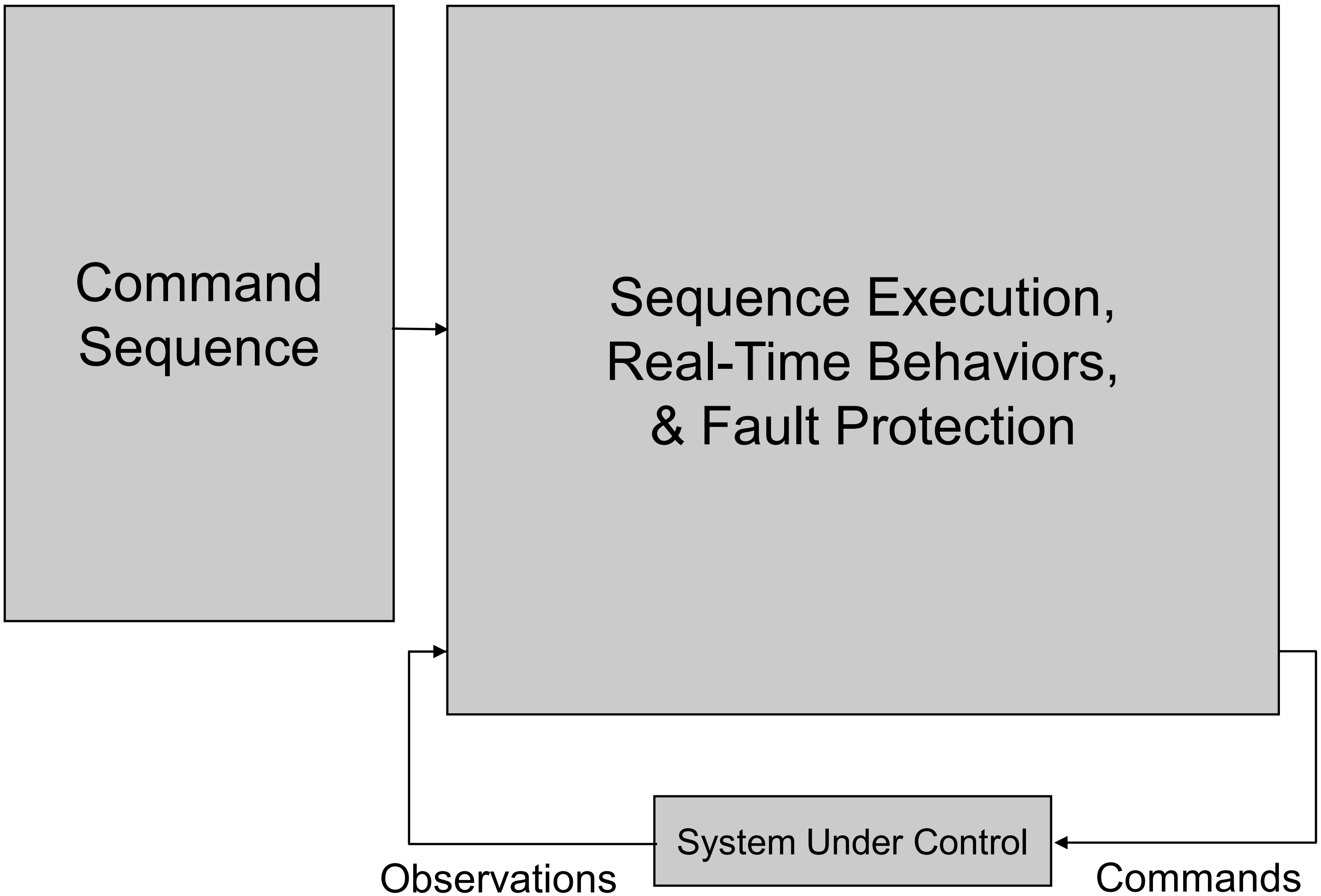
Model-based programs interact with the system's (hidden) state directly:

- Read state
- Set state

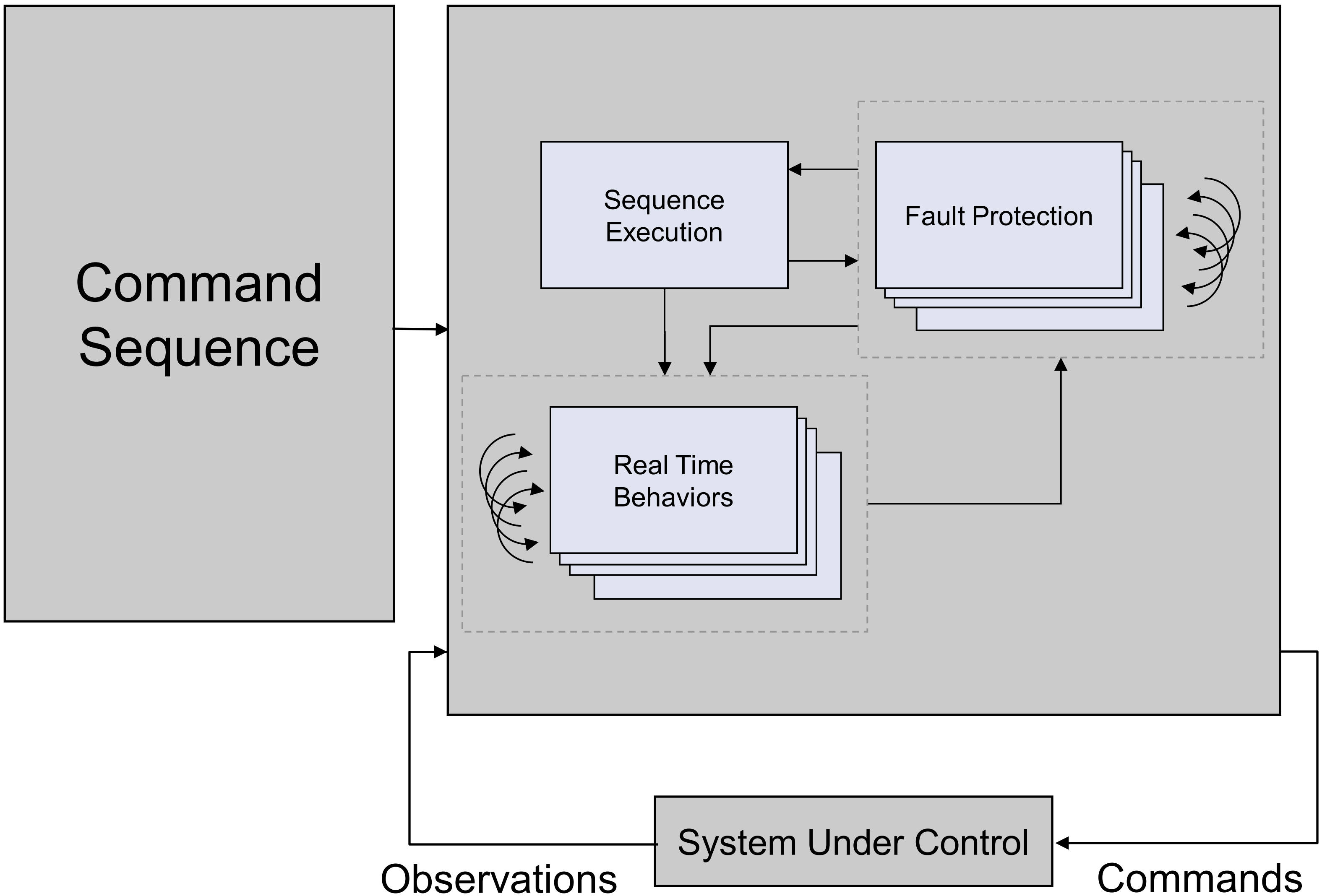


Model-based Executives automatically reason through interactions between states and sensors/actuators.

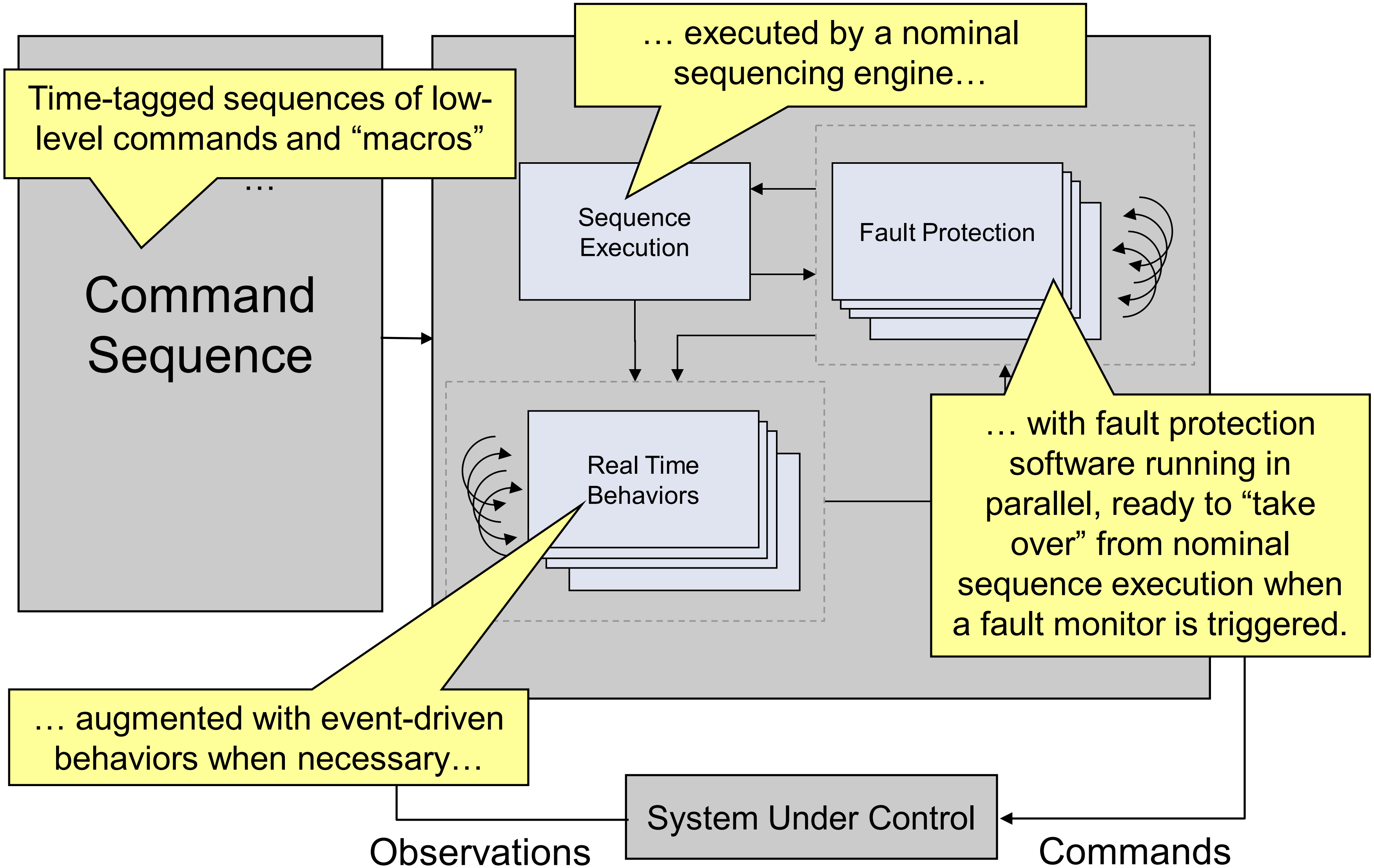
Typical Spacecraft Execution Architecture



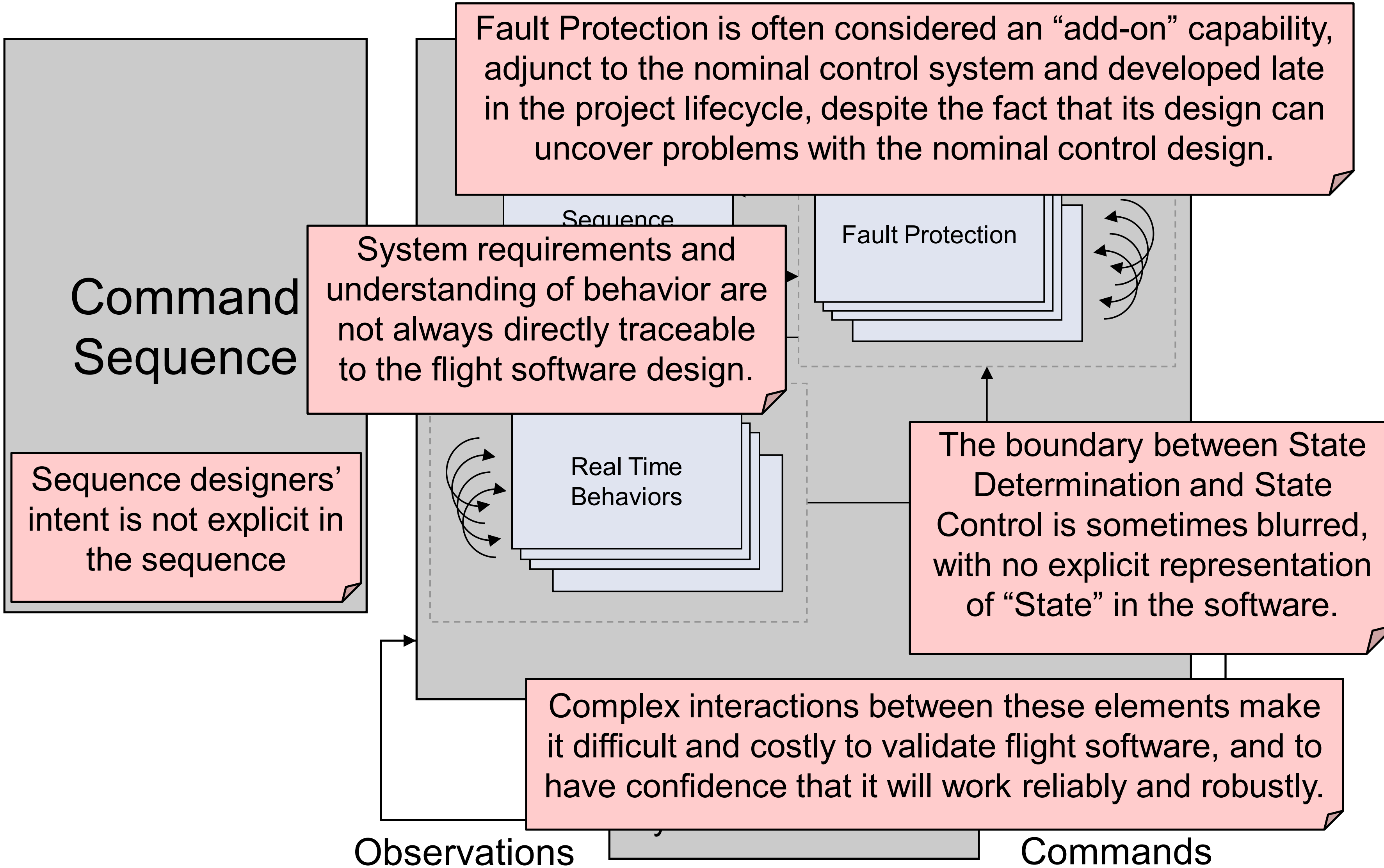
Typical Spacecraft Execution Architecture



Typical Spacecraft Execution Architecture



Limitations of the Typical Architecture



Fault Protection is often considered an “add-on” capability, adjunct to the nominal control system and developed late in the project lifecycle, despite the fact that its design can uncover problems with the nominal control design.

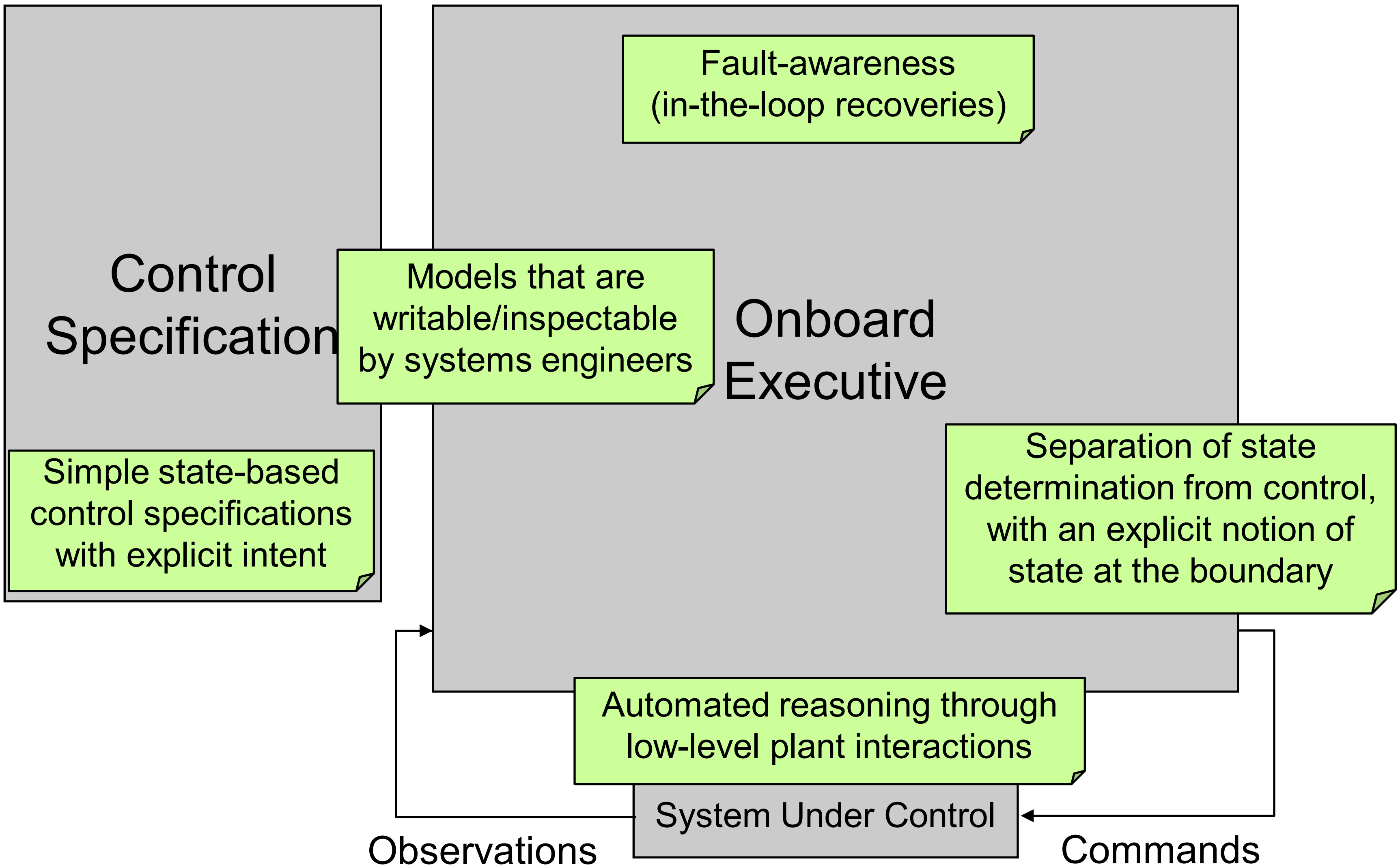
System requirements and understanding of behavior are not always directly traceable to the flight software design.

Sequence designers’ intent is not explicit in the sequence

The boundary between State Determination and State Control is sometimes blurred, with no explicit representation of “State” in the software.

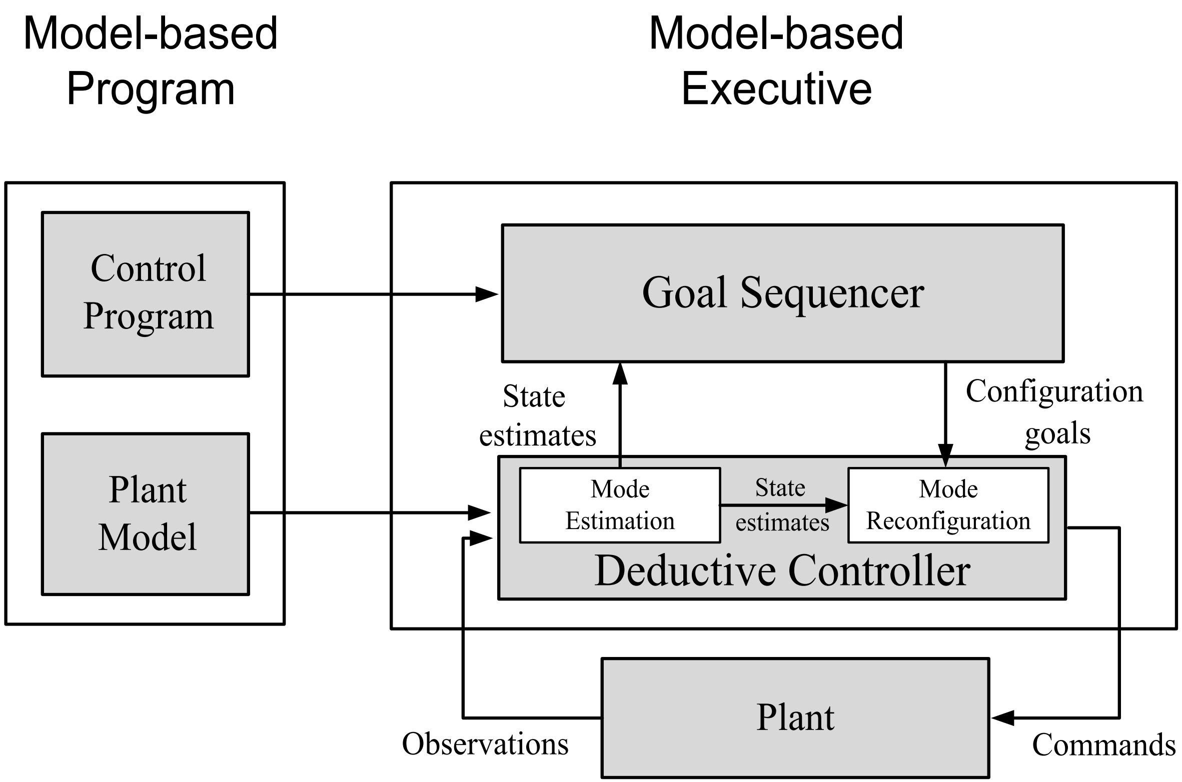
Complex interactions between these elements make it difficult and costly to validate flight software, and to have confidence that it will work reliably and robustly.

Desirable Architectural Features



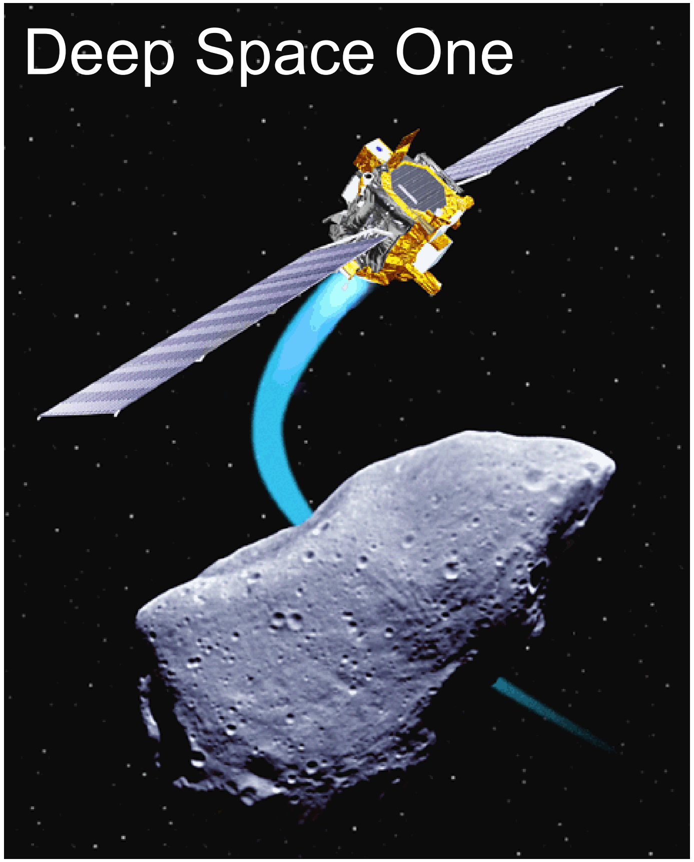
Titan Model-based Executive

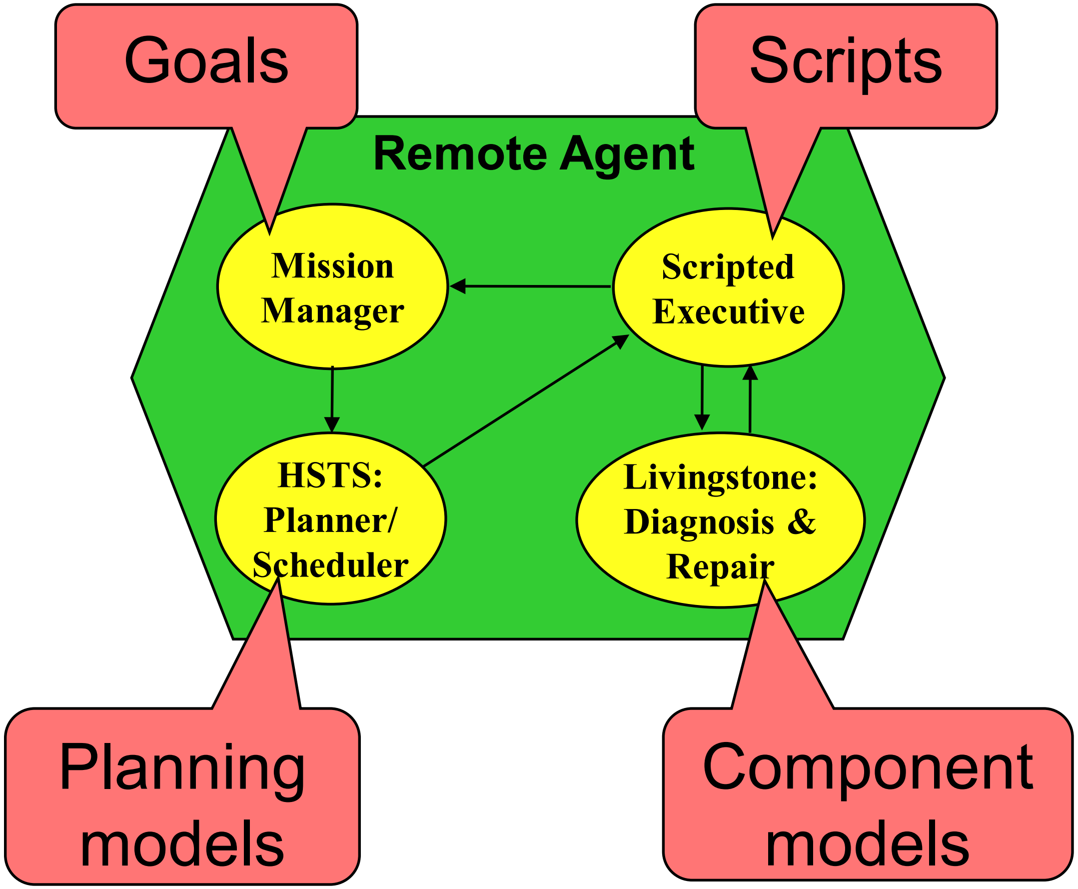
- **Control layer has flexibility in achieving goal**
- **Enables integration of tiered fault management capabilities**
- **Enables integration of state-of-the-art autonomy software**



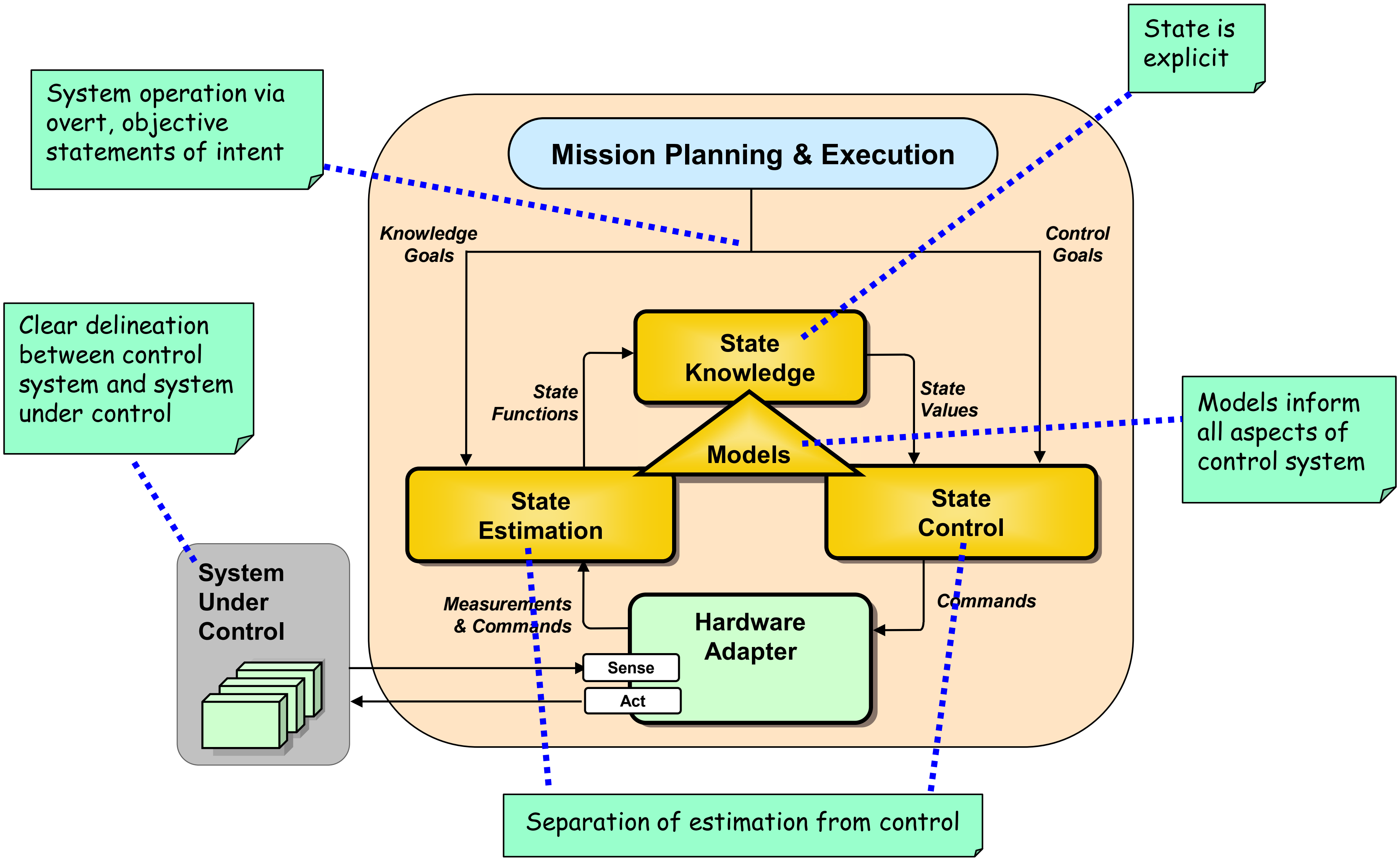
Williams, B.C., Ingham, M.D., Chung, S.H., and Elliott, P.H., "Model-based Programming of Intelligent Embedded Systems and Robotic Space Explorers", *Proceedings of the IEEE, Special Issue on Modeling and Design of Embedded Software*, Vol. 91, No. 1, Jan. 2003, pp. 212-237.

Remote Agent Experiment on DS-1





Mission Data System Reference Architecture



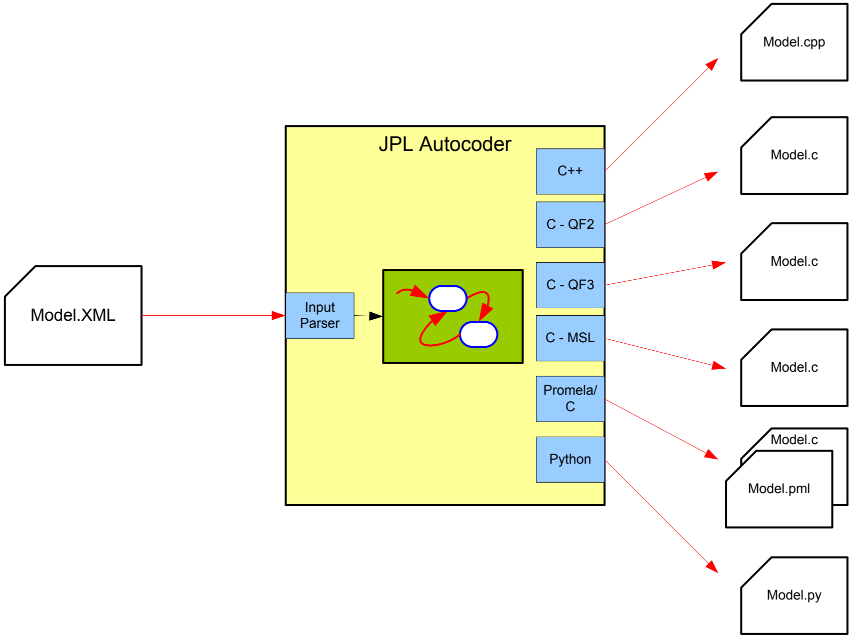
Challenges and Opportunities

- **Challenges:**
  - Closing the mid- to high-TRL gap
  - Must assure reliability ("bullet-proof" the implementation)
  - Changes the operational paradigm – need new tools, training
  - Cultural hurdles to acceptance of software technologies ("trust" issues)

- **Opportunities:**
  - Autonomy is an enabler for certain missions
  - Evidence of significant cost savings in operations (EO-1)
  - Model-based design lends itself well to development via MBSE methodologies
  - Once general-purpose reasoners have been validated, V&V reduced to mission-specific models
  - Amenability to formal V&V

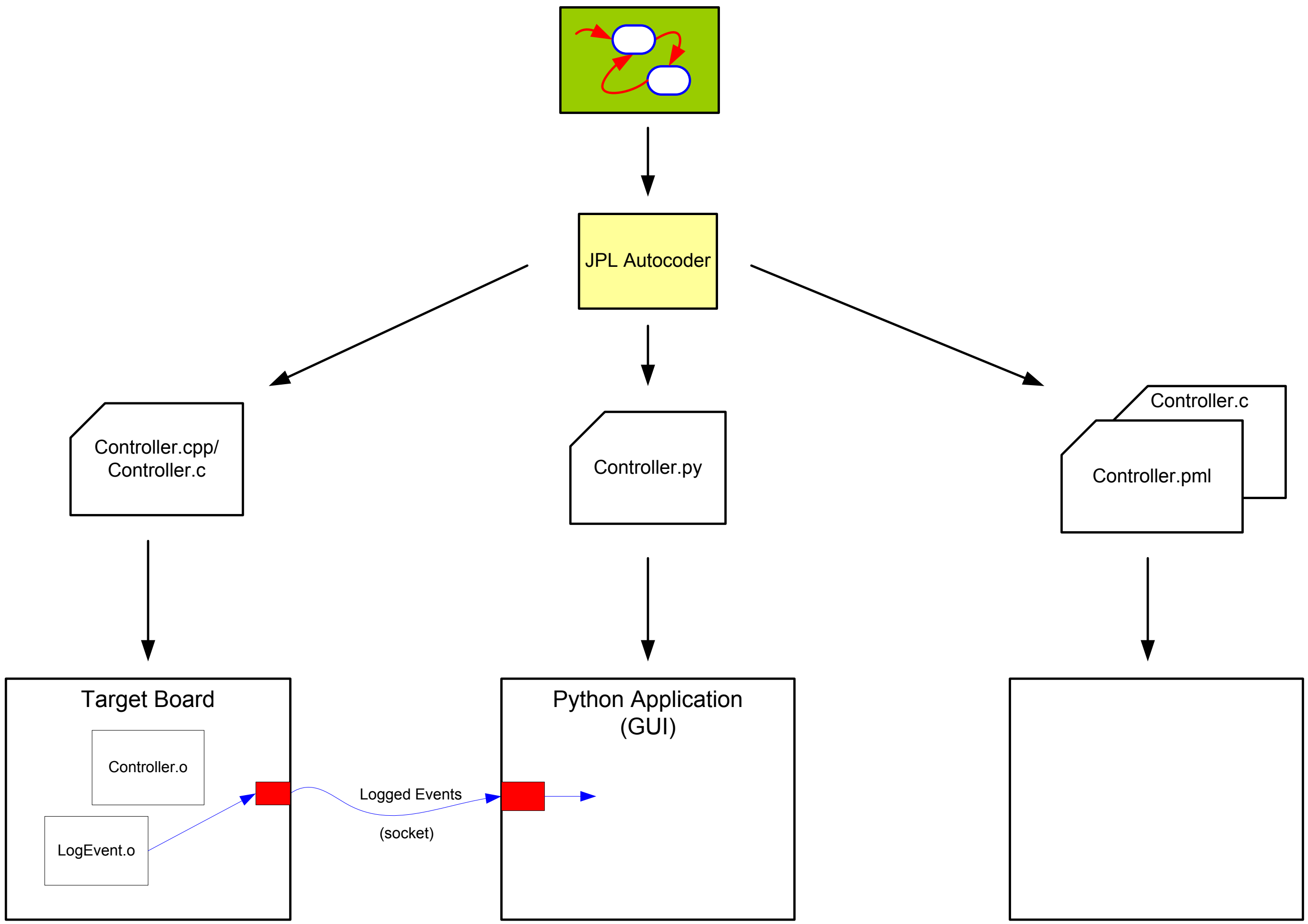
STAARS Auto-coder

- **UML Modeling**
  - Explicitly capture the intent of the requirements
  - Formally capture the behavior in a model
  - Create a crisp notion of ***state***
- **State-based Framework**
  - Supports the UML standard
  - Allows developers to think and work with higher constructs – states, events and transitions
- **Auto-coding**
  - Light-weight Java program
  - Reads in the Model which is stored in a non-proprietary data format (XML)
  - Converts the input model into an internal data structure
  - Has multiple back-ends to support different project requirements
- **Test harness**
  - Ability to run the model stand-alone – module test environment
- **Model checking**
  - Automatic generation of Verification models
  - Exhaustively explore the state-space of the model
  - Checks for various correctness properties within the model



STAARS Auto-coder

Source: Garth Watney, JPL



STAARS Process

STAARS Benefits

- **Lessened the gap between System and Software Engineering**
  - Formal specification of state behavior which can be implemented directly into flight software
  - Build rapid executable models for early prototype testing
- **Increased efficiency**
  - Software developers can greatly increase their output
- **Increased maintainability**
  - Rapid turn-around from specification changes to a software build
- **Increased reliability**
  - Fewer defects are introduced
  - Auto generated code based on a reliable statechart framework that conforms to the UML statechart semantics
- **Full control of the process**
  - Drawing tools can be swapped in and out
  - Autocoder can be customized for specific projects
    - Output in C or C++
    - Add more UML features – Deferred events, etc
    - Currently based on the Quantum Framework's Publish/Subscribe – but could be customized to be based on other Frameworks